UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Implementierung einer Warteschlange zur lastabhängigen Adaption der Bedienrate im Netzwerksimulator ns2

Bachelorarbeit

Leipzig, 31. Juli 2003

vorgelegt von:

Jens Geier
geb. am: 27.05. 1977

Studiengang Informatik

1. Kurzzusammenfassung

Ziel der Bachelorarbeit ist die Umsetzung und Erprobung eines Konzeptes zur Vermeidung von Überlastsituationen in paketvermittlenden Netzwerken. Hierzu wird ein Schedulingalgorithmus mit lastabhängiger Adaption der Bedienrate entwickelt – die Burst Shaping Queue. Dieser folgt dem Konzept der Bedienratenregulierung durch Verzögerung des Bedienzeitpunktes für Pakete. Die entwickelte Schedulingeigenschaft wird in die Klasse der Closed Loop Ansätze zur Überlaststeuerung eingeordnet und gegenüber Open Loop Ansätzen abgegrenzt. Zur Leistungsuntersuchung wurde der Algorithmus als Erweiterung des Netzwerksimulator ns2 umgesetzt. Verschiedene Testreihen belegen das gewünschte Verhalten der Implementierung. Die Testergebnisse werden im Hinblick auf die Verwendbarkeit des Ansatzes und zukünftige Entwicklungsmöglichkeiten analysiert und beurteilt.

Inhaltsverzeichnis

2. Motivation

Eine wesentliche Eigenschaft paketvermittelnder Netze ist statistisches Multiplexen, also die Möglichkeit, mehrere logische Verbindungen über einen Übertragungskanal zu realisieren. Eine kurzzeitige Übertragung von sehr vielen Daten und das damit sprunghaft ansteigende und schnell wieder abfallende Paketaufkommen, sogenannte Bursts oder Peaks, stellen für Vermittlungseinrichtung in Netzwerken extreme Ausnahmesituationen dar. Dabei wachsen Warteschlangen zeitweise oft so stark an, dass die Puffer in den Netzwerkkomponenten nicht ausreichen, um alle Pakete zwischenzuspeichern. Es entsteht eine Überlastsituation (Congestion). Um diese aufzulösen werden Pakete, für die keine Pufferkapazität mehr vorhanden ist, verworfen.
Das Transmission Control Protokoll (TCP) kann mit Paketverlusten, die durch Überlast im Netzwerk auftreten, gut umgehen. Die aktuelle Datenrate wird bei Paketverlust reduziert und dadurch eine Verringerung der Last im Netzwerk erreicht. Das Verwerfen von Paketen wird im Internet aber nicht nur zum Anzeigen sondern auch zum Vermeiden von Überlastsituationen benutzt. Durch Warteschlangenmechanismen, die in Abhängigkeit von der Auslastung einer Warteschlange sukzessive Pakete verwerfen, obwohl der Puffer noch nicht voll ist, lassen sich Überlastsituationen rechtzeitig ankündigen. Ein Beispiel für einen solchen Warteschlangenmechanismus ist Random Early Detection (RED)[2]. Das Verwerfen von Paketen ist im Moment die einzige breit eingesetzte Technik zur Vermeidung von Überlastsituationen [4]. Darüber hinaus existieren weitere Ansätze, die drohende Überlastsituationen durch Signalisierung in den Antwortpaketen anzeigen. Ein solches Vorgehen wird in [5] beschrieben.
Ziel der Aufgabenstellung der Bachelorarbeit ist die Umsetzung und Erprobung eines Konzeptes zur Vermeidung von Überlastsituationen in einzelnen Warteschlangen. Die Überlastung einer einzelnen Warteschlange soll durch eine bessere Ausnutzung von Warteschlangenkapazitäten entlang der Übertragungspfade und eine Veränderung des Verkehrsprofils erreicht werden. Server/Router, die nur wenige Pakete weiterleiten müssen, sollen diese bei konstanter Linkrate langsamer senden, als Router mit höherem Paketaufkommen.
Zu diesem Zweck wird die *Burst Shaping Queue* eingeführt. Anstatt wartende Pakete mit der Linkrate zu bedienen, werden diese mit einer vom Füllstand der Warteschlange abhängigen Rate bedient. Diese Rate ist nach oben durch die Linkrate beschränkt. Die untere Schranke sollte so gewählt werden, dass die Verzögerung im gesamten Netz kleiner als die von TCP angenommene Round-Trip-Zeit ist.
Im Rahmen der Arbeit wird eine generische Klasse für die Burst Shaping Queue im Netzwerksimulator (ns2) implementiert. Ziel der Arbeit sind erste generelle Aussagen zum Nutzen des Verfahrens. Der Einfluss verschiedener Schwell-Funktionen für die Veränderung der Bedienrate auf Verkehrsprofil und Verhalten der Warteschlange hin-

sichtlich der Überlastvermeidung auf dem Transportweg wird erörtert, an Hand von einfachen Simulationen getestet und bewertet. Zum Zwecke der allgemeinen Beurteilung wird das Verhalten von Burst Shaping Queues im Vergleich mit Standard Drop Tail Warteschlangen getestet und bewertet.

3. Problemanalyse

3.1. Einführung und allgemeines Vorgehen

Ziel der Arbeit ist die Umsetzung einer Warteschlange, die ihre Bediengeschwindigkeit in Abhängigkeit von der Warteschlangenlänge reguliert. Dieser Abschnitt bietet eine allgemeine Einführung in das Problem, diskutiert Lösungsansätze und definiert Begriffe.
Bei normalem Betrieb, d.h. bei *normaler Last*, ist die Ankunftsrate kleiner als die Bedienrate und ein Netzwerkknoten kann alle ankommenden Pakete sofort bearbeiten. Bei wachsender Ankunftsrate tritt ab einem bestimmten Zeitpunkt eine Gleichheit von Ankunfts- und Bedienrate ein. Dieser Zustand wird als *Grenzlast* bezeichnet. Weiteres Wachstum führt schließlich zu *Überlast*– die Paketankunftsrate in einem Knoten ist höher als die Bedienrate. Pakete müssen nun in der Warteschlange zwischengespeichert werden. Im ungünstigsten Fall führt eine Überlast zum Überlaufen von Warteschlangen und damit zu Paketverlusten. Ist Überlast auf einen kurzen Zeitraum begrenzt und herrscht am Ende der Überlastsituation wieder normale Last wird diese Situation im folgenden als *Burst* oder *Peak* bezeichnet. Ein Netzwerkknoten kann den Beginn einer Überlastsituation sehr einfach am Anwachsen seiner Warteschlange erkennen. Analog ist der Übergang von Überlast zu normaler Last bei konstanter Bedienrate mit einem Fallen der Warteschlangenlänge verbunden und kann so ebenfalls leicht erkannt werden.
Die *Abarbeitungsgeschwindigkeit* oder *Bedienrate* einer Warteschlange ist definiert als

$$Bedienrate = \frac{BedienteDatenmenge}{Bedienzeit}$$

Um die Bedienrate zu verändern, existieren grundsätzlich zwei Möglichkeiten:

1. Regeln der Datenmenge bei gleicher Bedienzeit
2. Regeln der Bedienzeit bei gleicher Datenmenge

Das Regulieren der Datenmenge bei gleicher Bedienzeit würde bedeuten, dass mehr bzw. weniger Daten in der gleichen Zeit befördert werden müssten. Das käme einer Manipulation der Linkrate gleich – ein Ansatz, der von vornherein aus zwei Gründen ausscheidet. Zum einen wird in der Aufgabenstellung explizit eine konstante gefordert, zum anderen wird die Linkrate eines Knotens von Netzwerkschichten beeinflusst, deren Dienste von Paketwarteschlangen genutzt werden (z.B. ISO/OSI Data Link Layer) und somit nicht direkt durch sie beeinflusst werden können. Linkraten sind daher in der Regel konstant. Eine Ausnahme bilden Netzwerktechnologien mit gemeinsam benutzten Übertragungsmedien (Shared Media) und stochastischen Zugriffsmethoden). Ihre Linkrate ist abhängig von der Anzahl der um das Medium konkurrierenden Stationen. Sie

ist nicht konstant, kann aber auf Grund des stochastischen Zugriffs ebenfalls nicht reguliert werden.
Es bleibt zum Regulieren der Bedienrate also nur die Beeinflussung der Bedienzeit für eine konstante Menge von Daten. Dies läßt sich über eine Manipulation des Bedienzeitpunktes für jedes zu sendende Paket erreichen. Pakete, die bedient werden, werden erst nach Ablauf einer bestimmten Zeit gesendet. Diese Zeit zwischen frühest möglicher Bedienung und dem eigentlichen Beginn der Bedienung wird im folgenden als *Verzögerung* bezeichnet. Die Summe aus Verzögerung und Bedienzeit für ein Paket, bei Bedienung mit Linkrate, ergibt dann die *effektive Bedienzeit* dieses Paketes. Ein Paket ist nun *in Bedienung*, wenn es entweder wartet (d.h. sich im Verzögerungszeitraum befindet) oder tatsächlich bedient (d.h. zum nächsten Knoten gesendet) wird.

Es ergibt sich nun:

$$Bedienrate = \frac{BedienteDatenmenge}{Zeiteinheit + Verzoegerung} = \frac{BedienteDatenmenge}{effektiveBedienzeit}$$

Bei maximaler Bedienrate, d.h. bei Bedienung mit Linkrate, wird sofort nach Abschluss der Bedienung eines Paketes mit der Bedienung des nächsten Paketes begonnen. Um die Bedienrate zu senken, wird noch eine bestimmte Zeit zwischen diesem frühesten Bedienzeitpunkt und dem eigentlichen Bedienzeitpunkt gewartet – also die Verzögerung und damit auch die effektive Bedienzeit erhöht. Die effektive Bedienzeit eines Paketes bestimmt die *aktuelle Bediengeschwindigkeit* der Warteschlange. Da die Verzögerung abhängig von der Länge der Queue sein soll, muss sie jedesmal, wenn sich die Länge der Warteschlange ändert, neu berechnet werden. Ist die Verzögerung gleich null wird die Queue mit Linkrate, also *maximaler Bedienrate*, abgearbeitet. Durch die Begrenzung der Verzögerung nach oben wird eine *minimale Bedienrate* der Warteschlange festgelegt.
Die Länge der Warteschlange ändert sich nur, wenn entweder ein neues Paket eintrifft und in die Queue eingestellt wird oder ein Paket bedient worden ist und die Queue verlässt. Auf diese beiden Ereignisse muss ein Algorithmus, der das Verhalten der Burst Shaping Queue umsetzt, reagieren. Der Programmablaufplan in Abbildung 3.1 zeigt einen ersten, allgemeinen Algorithmenentwurf um ein solches Verhalten umzusetzen.

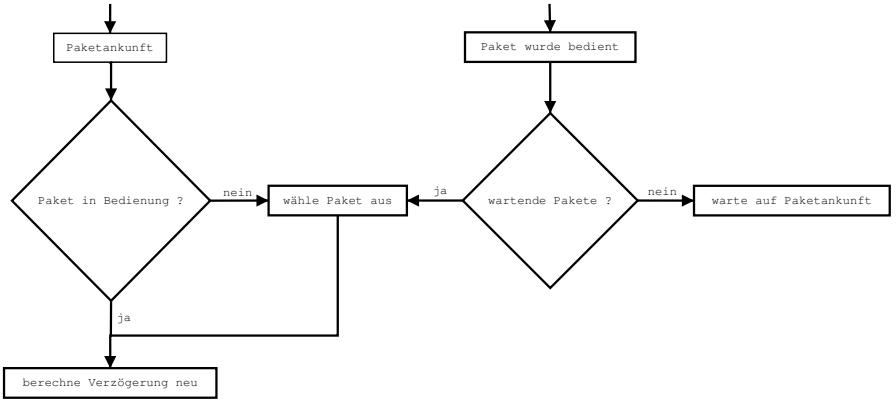

Abbildung 3.1.: Flußdiagramm Verzögerungsberechnung

Bei Eintreffen eines neuen Paketes ist zu prüfen, ob im Moment ein Paket bedient wird. Ist das der Fall, muss die Verzögerung für dieses Paket neu berechnet werden, da die Warteschlangenlänge erhöht wurde. Wird kein Paket bedient, muss mit der Bedienung des nächsten Paketes begonnen werden. Dazu ist ein Paket auszuwählen und die Verzögerung zu berechnen. Ist die Queue mit der Bedienung eines Paketes fertig, muss geprüft werden, ob es wartende Pakete gibt. Wenn das zutrifft, muss das nächste Paket bedient werden. Dazu wird ein Paket ausgewählt und die Verzögerung neu bestimmt. Wenn sich kein Paket mehr in der Warteschlange befindet, wartet die Queue auf das Eintreffen eines neuen Paketes.
Die Länge der Warteschlange ist nach oben durch die Kapazität des Warteschlangenpuffers beschränkt. Ihre untere Schranke stellt eine leere Warteschlange, also eine Warteschlange der Länge null dar. Innerhalb dieses Intervalls ist es von Vorteil, einen *Schwellwert* für das Erreichen der maximalen Bediengeschwindigkeit einzuführen. Bei wachsender Warteschlange würde so, ausgehend von der maximalen Verzögerung, die Verzögerung immer weiter reduziert und bei Erreichen des Schwellwerts der Warteschlangenlänge schließlich null annehmen. So kann schon vor dem Erreichen der maximalen Warteschlangenlänge die Warteschlange mit Linkrate bedient werden. Bei einem Schwellwert von null würde die Warteschlange immer mit Linkrate bedient und der Paketstrom nicht verändert werden.
Die Auswahl eines Paketes aus der Warteschlange folgt der Bedienstrategie der Queue (DropTail, Round Robin, Fair Queuing usw.) und ist somit unabhängig von der *Burst Shaping Eigenschaft*, die die Regulierung der Bediengeschwindigkeit in Abhängigkeit von der Wartreschlangenlänge beschreibt. Daher existiert „die“ Burst Shaping Queue, so wie es der Titel dieser Arbeit es vermuten lässt, genaugenommen nicht. Vielmehr existieren Bedienstrategien für Warteschlangen, die Burst Shaping Eigenschaften haben können. Zur grundsätzlichen Bewertung der Burst Shaping Eigenschaft wird im folgenden eine Kombination aus einfacher Drop Tail (FIFO) Bedienstrategie und Burst Shaping Eigenschaft als Burst Shaping Queue bezeichnet. Andere Kombinationen sind möglich, würden aber den Rahmen dieser Arbeit sprengen. Eine Annahme von Drop Tail Bedienstrategie ist sinnvoll, da sie leicht zu verstehen ist und trotzdem eine Bewertung des Burst Shaping Ansatzes erlaubt. Alle anderen einfachen Bedienstrategien, die sich mit einer günstigeren Paketauswahl beschäftigen, wie Round Robin oder Fair Queingm sind komplexere Spezialisierung dieser Bedienstrategie und können daher vernachlässigt werden. RED ist dem Burst Shaping Ansatz in dem Gesichtspunkt ähnlich, dass hier ebenfalls das Paketaufkommen zur Vermeidung von Überlast gezielt geregelt wird. Der Unterschied besteht vereinfacht gesagt darin, dass RED versucht, seine Ankunftsrate zu regulieren, während die Burst Shaping Eigenschaft die Bedienrate beeinflusst. Offen ist, ob sich beide Ansätze bei Kombination sinnvoll ergänzen würden. Um diese Frage zu beantworten, ist eine vorherige grundsätzliche Betrachtung des Burst Shaping Ansatzes notwendig, die in dieser Arbeit erfolgen soll.

3.2. Abschätzung von Grenzen und Auswirkungen des Ansatzes

Zur optimalen Überlastvermeidung in Netzen wird unter anderem in [6] Traffic Shaping vorgeschlagen. Dieser Ansatz, der genauso wie die Burst Shaping Queue Datenraten regelt, ist allerdings weitaus leistungsfähiger als diese. Trafffic Shaping ist ein Open Loop Ansatz und basiert auf der Aushandlung von Verkehrsmustern zwischen Teilnetzen und Benutzern. Diese sogenanten Flussspezifikationen legen die Eigenschaften eines Datenflusses genau fest. Traffic Shaping geht dabei von einer „bekannten“ Umgebung aus – es kennt alle wichtigen Parameter des Netzwerkes und trifft auf Basis dieser seine Entscheidungen. Solange sich Benutzer an diese Aushandlung halten, ist Überlast sehr unwahrscheinlich da das Netzwerk nicht mehr als die vorhandenen Ressourcen vergibt. Durch eine solche kontrollierte Vergabe vorhandener Ressourcen kann Überlast prinzipiell nicht entstehen. Allerdings ist ein solches Verfahren sehr aufwendig und resultiert in einem hohen technischen Aufwand für die einzelnen Netzknoten. Praktisch findet Traffic Shaping bei ATM-basierten Netzen Verwendung.
Im Gegensatz dazu ist die Burst Shaping Queue ein Closed Loop Ansatz; sie reagiert auf entstehende Überlast und verhindert sie nicht schon von vornherein. Die Burst Shaping Queue überwacht ständig die aktuelle Länge ihrer Warteschlange und die aktuelle Paketankunftsrate. Die Entwicklung dieser Werte über einen gewissen Zeitraum können weitere Messgrössen sein
Darüber hinaus nutzt die Burst Shaping Queue keine weiteren Informationen, weder über die Eigenschaften der in den Paketen enthaltenen Datenflüsse noch über Füllstände und Bedienraten von benachbarten Warteschlangen. Auf Grund ihres Funktionsprinzips hat sie keine Informationen über die Gesamtsituation des Netzwerkes – die Burst Shaping Queue kennt sozusagen nur „sich selbst“. Ihre Entscheidungen basieren auf unvollständigen Informationen über die Situation im gesamten Netzwerk, die aus den Messwerten der Parameter der eigenen Warteschlange gewonnen werden. Sie kann also nur in einem begrenzten Rahmen die Eigenschaften des Netzwerkes positiv beeinflussen. Ein grundsätzliches Vermeiden von Überlast ist schon durch den Closed-Loop Ansatz unmöglich [6].
Die Burst Shaping Queue verändert den Paketstrom, sobald ihre eigene Warteschlange wächst. In dieser Überlastsituation steigert sie ihre Bedienrate mit weiter wachsender Warteschlangenlänge langsam. Dadurch wird das Wachstum der Warteschlange bei einer konstant hohen Ankunftsrate nicht sofort bestmöglich, d.h. durch ihre physisches Leistungsmaximum, begrenzt. Erst bei Erreichen des Schwellwertes bedient die Queue mit maximaler Geschwindigkeit und schränkt so weiteres Warteschlangenwachstum bestmöglich ein. Wenn die Überlastsituation vorüber ist, also die Warteschlangenlänge wieder abnimmt und unter den Schwellwert fällt, veringert die Burst Shaping Queue ihre Bediengeschwindigkeit bei weiter abnehmender Warteschlängenlänge wieder. Die Burst Shaping Queue nutzt durch dieses Verhalten die Puffergröße ihrer eigenen Warteschlange als Zwischenspeicher, um eventuell nachfolgende Warteschlangen mit eventuell geringerer Linkrate vor Überlast zu schützen.
Durch die Regulierung der Bedienrate wird aber nicht nur die Pufferkapazität der Queue besser genutzt. Durch die Einführung der Verzögerung wird auch die Bedienrate bei

Warteschlangenabbau gesenkt – der Burst wird sozusagen „geglättet“. Nachfolgende Warteschlangen haben so weniger oder im Optimum sogar keine Überlast mehr zu bewältigen. Im Ausgleich dazu entsteht eine länger anhaltende, relativ konstante Normallast. Der Grad der Glättung kann über die Parameter der Burst Shaping Queue beeinflusst werden.
Im Abschnitt 3.1 wurde definiert, dass im Falle von Überlast die Ankunftsrate einer Warteschlange ihre Bedienrate übersteigt. Diese Aussage gilt prinzipiell auch für Burst Shaping Queues, muss in diesem Kontext aber noch näher erläutert werden. Eine normale Warteschlange bedient ankommende Pakete in der Regel mit ihrer Linkrate. „Überlast“ bezeichnet dann bei einer solchen Warteschlange eine tatsächliche Überbelastung, also das Überschreiten ihrer physischen Leistungsgrenze. Eine Burst Shaping Queue kann im Gegensatz dazu ihre Bediengeschwindigkeit zwischen null und der Linkrate beliebig regeln. Daher kann in einer Burst Shaping Queue Überlast nach obiger Definition schon bei Ankunftsraten, die kleiner als die Linkrate sind, entstehen. Folglich ist *Überlast* im Kontext einer Burst Shaping Queue nicht zwingend eine Überlastung von physischen Ressourcen, sondern bezeichnet allgemein das Anwachsen der Warteschlange. Eine niedrige Bedienrate bei Normallast kann so benutzt werden um auf das Profil des Datenverkehrs Einfluss zu nehmen, ohne das eine tatsächliche Überlast entsteht. Entscheidender Faktor ist dabei die in der Burst Shaping Queue festgelegte Bedienrate für eine nicht-wachsende Warteschlange, also die Verzögerung für Pakete, die sofort bedient werden können. Im folgenden wird sie mit *initialer* Verzögerung bezeichnet. Eine initiale Verzögerung von null bewirkt ein Verhalten ähnlich normalen Warteschlangen – die Warteschlange wächst nur in Folge tatsächlicher Überlast. Man kann durch Wahl der initialen Verzögerung und der daraus resultierenden *initialen Bediengeschwindigkeit* bestimmen, ab welcher Ankunftsgeschwindigkeit die Burst Shaping Queue beginnt, auf den Verkehr Einfluss zu nehmen. Solange diese unterschritten wird verhält sich eine Burst Shaping Queue wie eine Drop Tail Queue und beeinflusst den Datenverkehr nicht aktiv.

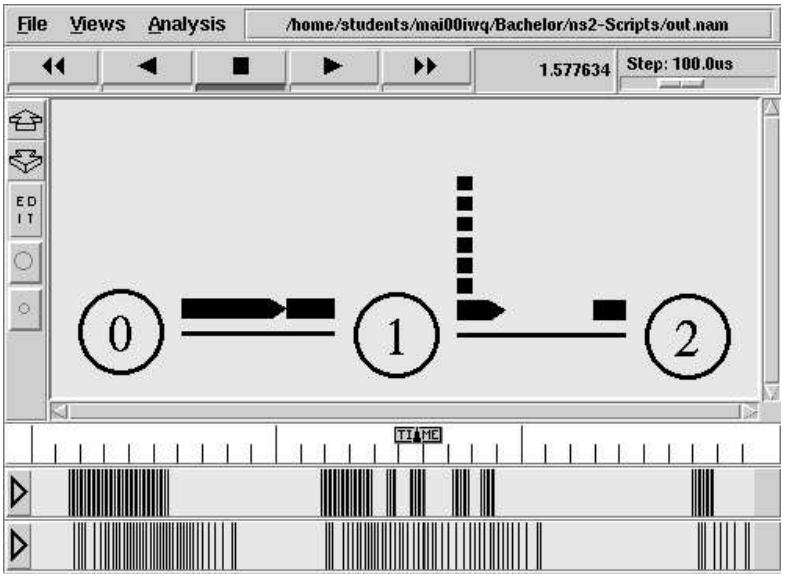

Abbildung 3.2.: Wirkungsweise der Burst Shaping Queue

Die Abbildung 3.2 illustriert den Einfluss einer Burst Shaping Queue auf Paketströme an einer sehr einfachen Konfiguration. Gezeigt wird eine im *ns2* implementierte Burst Shaping Queue am Knoten 1, die auf den Paketstrom zwischen Knoten 0 und 2 wirkt. Dieser Datenstrom enthält viele Bursts. Den oberen Teil der Abbildung bildet die Verkehrsvisualisierung in der Knoten als Kreise und Datenpakete als schwarze Pfeile dargestellt sind. An den Abständen der Pakete ist deutlich die Änderung der Bedienrate durch die Verzögerung einzelner Pakete zu erkennen.Die unteren beiden Balkeniagramme zeigen den zeitlichen Verlauf des Paketflusses. Jeder senkrechte Balken markiert dabei ein Paket, die Zeitachse verläuft von links nach rechts. Das obere Diagramm zeigt den Eingangsdatenstrom der Burst Shaping Queue, unten ist der Ausgangsdatenstrom dargestellt. Beide zeigen sehr deutlich wie die Burst Shaping Queue den Datenverkehr zwischenpuffert und mit niedriger Bedienrate wieder abgibt. Die Veränderung der Datenrate ist auch hier an den veränderten Paketzwischenräumen gut zu erkennen. Die Lastspitzen im mittleren Bereich werden vollständig in eine einzige, relativ konstante Übertragung mit niedrigerer Datenrate gewandelt.
Dieses gewünschte Verhalten zeigt die Burst Shaping Queue nur, wenn die Überlastsituation zeitlich eng begrenzt ist. Auf längere Peaks reagiert die Burst Shaping Queue zweistufig. Solange die Warteschlange wächst, wird bis zum Erreichen des Schwellwertes die Bedienrate bis auf die Linkrate gesteigert. Wird der Schwellwert der Warteschlange überschritten, bedient die Queue mit maximaler Geschwindigkeit. Sie nimmt bei weiterem Wachstum nun das Verhalten einer DropTail Queue an und behält dieses bis der Schwellwert wieder unterschritten wird. Sollte die Warteschlange nach Überschreiten des Schwellwertes immer noch wachsen und ihre Gesamtkapazität überschritten werden, beginnt die Queue mit Load Shedding. Genauso wie eine Drop Tail Queue verwirft sie als letztes Mittel zur Überlastauflösung alle Pakete, die nicht in ihren Puffer passen.
In Abbildung 3.2 ist zu sehen, dass die Burst Shaping Queue im Datenverkehr cha-

rakteristische Spuren hinterläßt. Sobald sie ihre Warteschlange komplett leert und keine Pakete in die Schlange eingestellt werden, folgen in ihrem Ausgangsdatenstrom die letzten beiden Pakete direkt aufeinander. Dieses Verhalten ist in der Implementierung und der Wahl der initialen Verzögerung in diesem Beispiel begründet. Wenn das vorletzte Paket in der Schlange bedient ist und das letzte Paket in Bedienung kommt, ändert sich die Warteschlangenlänge auf null und das aktuell zu bedienende Paket erhält die initiale Verzögerung. Da für die initiale Verzögerung hier null gewählt wurde wird es direkt nach dem vorletzten Paket gesendet. Die Implementierung dieser Warteschlange bezieht nur die aktuelle Warteschlangenlänge in ihre Verzögerungsberechnung ein. Sie kann so nicht zwischen zunehmenden und abnehmenden Warteschlangen unterscheiden. Daher behandelt sie das letzte Paket bei fallender Warteschlangenlänge genauso wie Pakete, die kein Wachstum der Warteschlange verursachen: sie werden mit initialer Verzögerung zurückgehalten und dann bedient. Die Erkennung einer solchen Situation und Vermeidung der geschilderten Spuren im Datenverkehr sind grundsätzlich möglich, wurden hier aber nicht umgesetzt.
Der erwartete Nutzen der Burst Shaping Queue ist also in engen Grenzen und im Vergleich mit Drop Tail Warteschlangen zu sehen. Eine Verbesserung der Netzwerkeigenschaften und damit ein Nutzen der Burst Shaping Queue ist erreicht, wenn die Burst Shaping Queue sich bei Peaks im Datenverkehr besser verhält als Drop Tail Warteschlangen. Die Burst Shaping Queue kann nachfolgende Warteschlangen vor kurzzeitiger Überlast schützen – aber nur, wenn sie selber überlastet wird, d.h. ihre Warteschlange wächst. Sie hat keine Möglichkeit, Paketverluste komplett zu vermeiden oder zu kompensieren, sie kann nur den Zeitpunkt des Eintretens von Verlusten bei Überlast durch Nutzung ihrer Warteschlange als Puffer hinauszögern . Darüber hinaus ist die BurstShaping Queue in der Lage, Datenströme zu glätten. Ein geglätteter Datenstrom weist keine Peaks mehr auf und verhindert so wirksam das Auftreten von kurzzeitiger Überlast.

Die Burst Shapig Queue hat nicht nur die gewünschten positiven Auswirkungen auf den Datenverkehr. Durch die Einführung einer Verzögerung für Pakete – also der Beeinflussung des zeitlichen Abstandes zwischen zwei Paketen – werden außer der Bedienrate noch zwei wesentliche Netzwerkkenngrößen, manipuliert: Delay und Jitter. Beide hängen direkt von der Verzögerung ab. *Delay* bezeichnet dabei die Übertragungszeit für ein Paket von Ende zu Ende über das Netzwerk hinweg. Vereinfacht gesagt ist das Delay die Summe der Verzögerungen, Wartezeiten und Bedienzeiten in allen Warteschlangen auf dem Übertragungspfad, zuzüglich der (vernachlässigbar kleinen) Zeit für die Übertragung auf den Medien zwischen den Knoten. Das *Jitter* ist ein Maß für die Schwankung in den zeitlichen Abständen von Paketen eines Datenstromes. Bei konstant Abständen ist der Jitter gleich null.
Beide Kenngrößen werden durch alle beteiligten Warteschlangen des Übertragungsweges beeinflusst und geraten damit auch in den Einfluss des BurstShaping Ansatzes. Die Einführung einer zusätzlichen Verzögerung für jedes Paket zur Senkung der Bedienrate läßt das Delay der Verbindung steigen. Sollte der Datenstrom viele Bursts enthalten, werden Warteschlangen oft wachsen und abgebaut werden. Diese Schwankung hat eine intensive Regelung der Bedienrate und damit das Erzeugen von vielen, unterschiedlich langen Paketverzögerungen zur Folge. Der Datenstrom enthält so im ungünstigsten Fall einen zusätzlichen Jitter.

Insbesondere bei der Übertragung von isochronen Daten wie Audio/Videostreams sind ein niedriges Delay und ein kleines Jitter von fundamentaler Bedeutung. Eine intensive Veränderung dieser Kenngrössen durch Burst Shaping Queues würde zu schlechterer Verbindungsqualität bzw. zu Verbindungsabbruch führen. Aber auch andere, nicht zeitkritische Übertragungen können in Mitleidenschaft gezogen werden. Beispielsweise kann ein zu hohes Delay zum Ablauf von Timeouts führen und die Verbindung wird fälschlicherweise als abgebrochen angesehen.
Die negativen Auswirkungen der Burst Shaping Queue auf Delay und Jitter sind trotz den oben geschilderten relativ drastischen Konsequenzen aus folgenden Gründen akzeptabel. Zum einen werden Delay und Jitter von der Verzögerung beeinflusst, die sich je nach Warteschlangenlänge zwischen einem vorgegebenen Minimum und einem ebenfalls vorgegebenen Maximum bewegt. Eine gute Wahl dieser Parameter kann die maximale Schwankung von Delay und Jitter sinnvoll einschränken und negative Auswirkungen durch die Burst Shaping Queue praktisch vernachlässigbar machen. Zum anderen beeinflusst die Burst Shaping Queue den Datenverkehr nur im Falle von Überlast und auch dann nur, wenn die Warteschlangenlänge kleiner als der Schwellwert ist. In einem solchen Fall hat eine Verschlechterung der Verbindungsqualität ohne Abriss des Datenstromes jedoch weniger Konsequenzen als der Verlust von Paketen und der damit unterbrochene Datenstrom. Bei nichtwachsenden Warteschlangen oder Warteschlangenlängen jenseits des Schwellwertes werden Delay und Jitter nur gering (durch die initiale Verzögerung) oder gar nicht beeinflusst.
Die Wahl von maximaler und initialer Verzögerung, der maximalen Warteschlangenkapazität, des Schwellwertes sowie die Relation von Bedienrate/ Verzögerung und aktueller Warteschlangenlänge sind von fundamentaler Bedeutung für den Nutzen des Konzeptes. Ihre Wahl bestimmt den Einfluss der Burst Shaping Queue auf den Datenverkehr. Für eine erste Bewertung des Ansatzes ist es sinnvoll, geeignete Werte für diese Parameter experimentiell zu bestimmen. Über das Verhalten der Bedienrate in Abhängigkeit von der Warteschlangenlänge können präzisere mathematische Aussagen gemacht werden die zur Bestimmung von Funktionen genutzt werden. Eine Beurteilung geeigneter Funktionen kann aber wiederum nur experimentiell vorgenommen werden.

3.3. Mathematisch - Formale Grundlagen

Um das Verhalten der Burst Shaping Queue umzusetzen und den Ansatz zu bewerten, ist eine präzise Beschreibung der relevanten Eigenschaften der Warteschlange von Vorteil. Desweiteren erleichtern formale mathematische Formulierungen das Finden von Funktionen, die Warteschlangenlänge und Bedienrate/Verzögerung in Relation setzen. Aus diesem Grund werden in diesem Abschnitt Eigenschaften der Queue mathematisch definiert und darauf aufbauend Zusammenhänge formalisiert.
Für die Warteschlange werden nun folgende Definitionen festgelegt:

- **Def:** r_l Linkrate der Queue, $r_l > 0$,$r_l = const.$
- **Def:** l_{max} maximale Queuelänge, $l_{max} > 0$, $l_{max} = const.$
- **Def:** r_b effektive Bedienrate für das aktuelle Paket, $0 \leq r_b \leq r_l$
- **Def:** l Länge der Queue zum Zeitpunkt der Neuberechnung, $0 \leq l \leq l_{max}$
- **Def:** s Schwellwert der Queue für maximale Bedienrate, $0 \leq s \leq l_{max}$
- **Def:** L Länge des aktuellen Paketes p in Bytes

Für das zeitliche Verhalten einzelner Pakete in der Queue werden definiert:

- **Def:** $t_b(r_l) = \frac{L}{r_l}$, Bedienzeit des aktuellen Paketes,
- **Def:** t_{dinit}, initiale Verzögerung durch die Queue, $0 \leq t_{dinit}$
- **Def:** t_{dmax}, maximale Verzögerung durch die Queue, $t_{dinit} \leq t_{dmax}$
- **Def:** $t_d = f(l, s, t_{dinit}, t_{dmax})$, Verzögerung als parametrisierte Funktion von l

Aus diesen Definitionen lassen sich nun formal zwei wichtige Zusammenhänge, die das Verhalten der Queue charakterisieren, formulieren:

1. **E**ffektive Bedienrate für aktuelles Paket
   $r_b = \frac{L}{(t_b(r_l)+t_d} = \frac{L}{f(l,s,t_{dinit},t_{dmax})}$
2. **B**edienung mit Linkrate also oberer Schranke für r_b
   $r_b = r_l \Leftrightarrow t_d = f(l, s, t_{dinit}, t_{dmax}) = 0$

Es gilt $t_b(r_l) = L/r_l$ und sowohl L als auch r_l sind bekannt. So reduziert sich das Problem der Beschreibung des Warteschlangenverhaltens auf das Finden von geeigneten Funktionen $f_i(l, s, t_{dinit}, t_{dmax})$, die im folgenden *Verzögerungsfunktionen* genannt werden.

Diese haben folgende gemeinsame mathematische Eigenschaften:

- **o**bere Schranke
  - $0 < f_i(l, s, t_{dinit}, t_{dmax}) <= t_{dmax}, l > 0$
    (t_{dmax} so gewählt, das TCP-RTT im ganzen Netz unterschritten wird)
- **u**ntere Schranke
  - $f_i(l, s, t_{dmax}) = 0 \Leftrightarrow s \leq l \leq l_{max}$
    (Bedienung mit Linkrate beim Überschreiten der Schwelle s)
- **M**onotonie
  - $0 \leq l_1 < l_2 \leq_{max} \Leftrightarrow f_i(l_1, s, t_{dinit}, t_{dmax}) \geq f(l_2, s, t_{dinit}, t_{dmax})$
    (Garantie des Ansteigens der effektiven Bedienrate in den geforderten Grenzen)
- **N**eutrales Verhalten gegenüber nichtwachsender Warteschlange
  - $f_i(0, s, t_{dmax}) = t_{dinit}$

Die Abhängigkeit der Verzögerungsfunktionen von den Parametern s, t_{dmax} und t_{dinit} erschwert das Bestimmen geeigneter Funktionen. Da die Parameter s und t_{dmax} aber Definitions- und Wertebereich auf je ein abgeschlossenes Intervall begrenzen, lassen sich beide durch geeignete Hilfsfunktionen auf [0,1] normalisieren. t_{dinit} ist nur Funktionswert des Argumentes $l = 0$ und kann daher in den folgenden Betrachtungen vernachlässigt werden.

1. **Def:** l_n normalisierte Queuelänge
2. **Def:** t_dn normalisierte Verzögerung
3. **T**ransformation von l („Hintransformation“)
   - $t_{hin} : [1, l_{max}] \rightarrow [0, 1]$ mit
   - $l_n = t_{hin}(l, s) = \begin{cases} \frac{l-1}{s-1}, l \leq s \\ 1, sonst \end{cases}$
4. **T**ransformation von t_{dn} („Rücktransformation“)
   - $t_{rueck} : [0, 1] \rightarrow [0, t_{dmax}]$ mit
   - $t_{rueck}(t_{dn}, t_{dmax}) = t_{d-normal} * t_{dmax}$

Für $f_i(l, s, t_{dinit}, t_{dmax}$ gilt dann unter Nutzung der Transformation:

$$f_i(l, s, t_{dinit}, t_{dmax}) = \begin{cases} t_{rueck}(g_i(t_{hin}(l, s)), t_{dmax}), 0 < l \leq s \\ t_{dinit}, l = 0 \end{cases}$$

Die verbliebenen zu bestimmenden *normalisierten Verzögerungsfunktionen* $g_i(l)$ haben dann folgende günstige Eigenschaften:

- **A**bbildungseigenschaft
  - $g : [0, 1] \rightarrow [0, 1]$
- **N**ullstelle
  - $g(0) = 1$
- **D**urchstoßpunkt
  - $g(1) = 0$
- **M**onotonie
  - $x_1 < x_2 \Leftrightarrow g(x_1) \geq g(x_2)$

3.4. Bestimmen geeigneter Verzögerungsfunktionen

Die im letzten Abschnitt beschriebenen Eigenschaften stellen einen Rahmen für potentielle Verzögerungsfunktionen $g_i(l)$ dar. Das Verhalten der Funktionen innerhalb dieses Rahmens ist für das spätere Verhalten der Queue von entscheidender Bedeutung. Ziel ist es, eine Funktion zu finden, die auf das Gesamtverhalten der Datenverbindungen möglichst günstigen Einfluss hat, d.h sie soll Überlast möglichst lange vermeiden und gleichzeitig die Netzwerkparameter Delay und Jitter möglichst wenig beeinflussen. Da diese drei Faktoren konkurrieren, soll die Zielfunktion einen möglichst guten Kompromis zwischen Verkehrsbeeinflussung und Delay- und Jitterverschlechterung bieten.
Als erste Funktion und gleichzeitig Ausgangspunkt für weitere Betrachtungen lässt sich eine lineare Verzögerungsfunktion bestimmen. Diese stellt Proportionalität zwischen Warteschlangenlänge und Verzögerung her.

- **l**ineare Verzögerungsfunktion
  - $g_0(l) = -x + 1$ (normalisiert)
  - $f_0(l, s, t_{dmax}) = -\frac{l-1}{s-1} * t_{dmax}$

Mit Hilfe von $g_0(l)$ lassen sich die $g_i(l)$ nun grob zwei Gruppen zuordnen:

- **F**unktionen $g_j(l)$, deren Graph oberhalb von $g_0(l)$ verläuft
  - $\forall x \in (0,1)(g_j(x) > g_0(x))$
- **F**unktionen $g_k(l)$, deren Graph unterhalb von $g_0(l)$ verläuft
  - $\forall x \in (0,1)(g_k(x) < g_0(x))$

Funktionen der Gruppe $g_j(l)$ reduzieren bei kleinen l die Verzögerung langsamer als $g_0(l)$, d.h. die Warteschlange wächst schneller. Sie streben bei großen l schneller eine kleine Verzögerung an als $g_0(l)$. Die $g_k(l)$ verhalten sich exakt umgekehrt: sie lassen im Vergleich zu $g_0(l)$ bei kleinen l die Verzögerung schneller fallen, d.h. die Warteschlange wächst langsamer, bei großen l dagegen langsamer. Darüber hinaus ist die Änderung der Verzögerung bei beiden nicht so gleichmäßig wie bei $g_0(l)$ – sie formen also den Datenverkehr nicht so gut.
Im Extrem approximieren die Funktionen $g_k(x)$ die Funktion g_{fifo} und $g_j(l)$ die Funktion $g_{delay}(l)$ mit

- $g_{fifo} = \begin{cases} t_{dn} = 0, l > 0 \\ t_{dn} = 1, l = 0 \end{cases}$
- $g_{delay} = \begin{cases} t_{dn} = 1, l \leq 0 \\ t_{dn} = 0, l = 1 \end{cases}$

Das Verhalten von $g_{fifo}(l)$ gleicht im Prinzip dem einer FIFO/ DropTail Queue – Pakete werden praktisch nicht verzögert und die Bedienrate somit nicht geregelt (unter Vernachlässigung des Falles $l = 0$). Dagegen verzögert $g_{delay}(l)$ jedes Paket bis zum Erreichen des Warteschlangenschwellwertes maximal. Sie resultiert in einer Warteschlange, die Pakete bis zum Erreichen des Schwellwertes staut und nicht wieder aus ihrem Puffer entlässt.
Beide Funktionen zeigen ein Verhalten, das durch die Implementierung nicht gewünscht ist und werden daher nicht eingesetzt. Sie dienen aber zusammen mit $g_0(l)$ zur groben Bestimmung des voraussichtlichen Verhaltens einer Verzögerungsfunktion. Wenn man den Verlauf einer Verzögerungsfunktion mit dem Verlauf der Funktionen $g_0(l)$, $g_{fifo}(l)$ und $g_{delay}(l)$ vergleicht, kann man Voraussagen über ihren Einfluss auf den Datenverkehr treffen. Je stärker eine Funktion $g_0(l)$ in ihrem Verlauf approximiert, desto intensiver wird sie den Datenverkehr formen. Je näher sie an einer der beiden anderen Funktionen liegt, desto weniger ausgeprägt wird diese Eigenschaft. Je stärker eine Funktion $g_{fifo}(l)$ approximiert, desto stärker wird sie sich wie eine Drop Tail Queue verhalten und Pakete schon bei geringen Warteschlangenlängen mit fast maximaler Geschwindigkeit bedienen. Im Kontrast dazu wird eine Funktion, je stärker sie g_{delay} approximiert, Pakete schon bei geringen Warteschlangenlängen relativ lange in ihrer Warteschlange behalten und diese auch nur sehr langsam bedienen. Es werden je zwei Vertreter aus den $g_j(l)$ und $g_k(l)$ mit unterschiedlichen Approximationseigenschaften an die Extrema bzw. $g_0(l)$ für die Tests ausgewählt:

- **q**uadratische Verzögerungsfunktion aus $g_j(l)$
  - $g_1(l) = 1 - x^2$ (normalisiert)
  - $f_1(l, s, t_{dmax}) = 1 - \frac{l-1}{s-1}^2 * t_{dmax}$
- **q**uadratische Verzögerungsfunktion aus $g_k(l)$
  - $g_2(l) = (x-1)^2$ (normalisiert)
  - $f_2(l, s, t_{dmax}) = (\frac{l-1}{s-1} - 1)^2 * t_{dmax}$
- **K**reis aus $g_j(l)$
  - $g_3(l) = \sqrt{1 - x^2}$ (normalisiert)
  - $f_3(l, s, t_{dmax}) = \sqrt{1 - \frac{l-1}{s-1}^2} * t_{dmax}$
- **K**reis aus $g_k(l)$
  - $g_4(l) = \sqrt{1 - (x-1)^2}$ (normalisiert)
  - $f_4(l, s, t_{dmax}) = \sqrt{1 - (\frac{l-1}{s-1} - 1)^2} * t_{dmax}$

Das Verhalten dieser Funktionen im Intervall $[0, 1]$ zeigt die Abbildung 3.3. Die Funk-

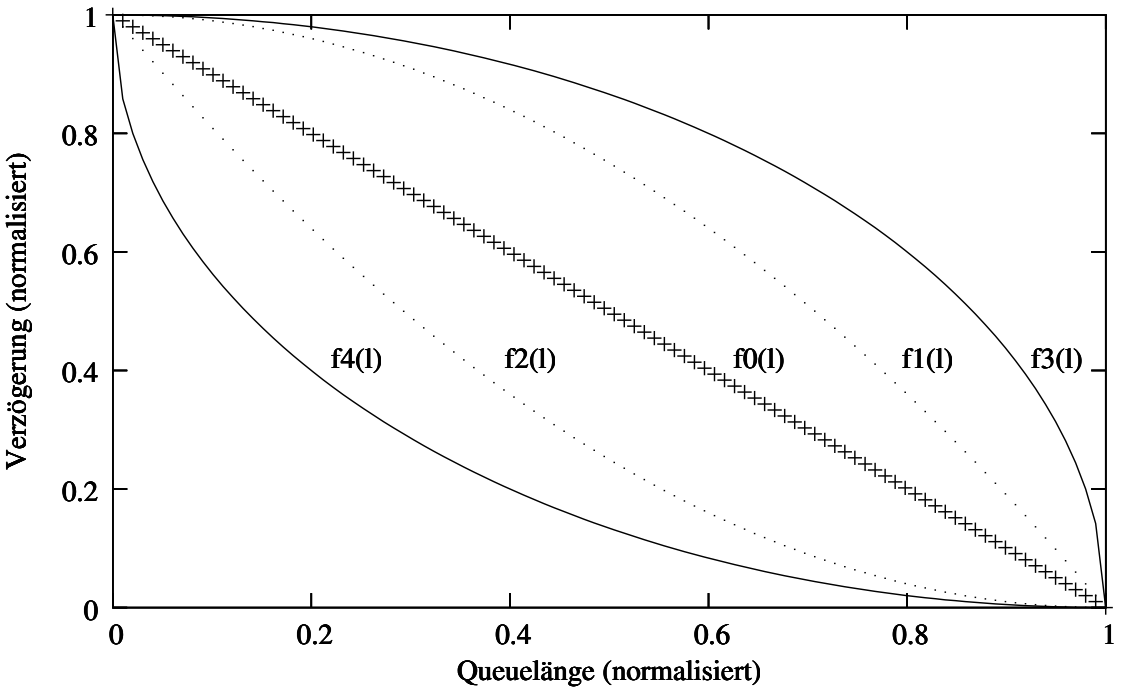

Abbildung 3.3.: Graphen der gewählten $g_i(l)$

tion $f_0(l)$ ist eine direkte Umsetzung der linearen Verzögerungsfunktion und dient als Basis für die Einschätzung des Verhaltens der anderen Funktionen. Ihre Glättungswirkung auf den Datenverkehr ist sehr gleichmässig und unabhängig von der Länge der Bursts. Ihre Warteschlange wird mit durchschnittlicher Geschwindigkeit wachsen.
Die Funktionen $f_2(l)$ und $f_4(l)$ werden dagegen weniger formend auf den Datenverkehr wirken. Bereits kleine Bursts werden von ihnen schnell bedient, also nur in geringerem Maß geformt als $f_0(l)$ dies täte. Dabei hat $f_4(l)$ eine noch weniger ausgeprägte Formungseigenschaft als $f_2(l)$. Ihre Warteschlangen werden aber dafür langsamer wachsen als die von $f_0(l)$, wobei die Warteschlange von $f_2(l)$ schneller wächst als die von $f_4(l)$. Verzögerungsfunktionen aus dieser Klasse sind auf Grund dieser Eigenschaften eher für das Ausfiltern sehr kurzer Bursts geeignet, da ihr Einfluss auf die Gestalt des Datenverkehrs mit wachsender Warteschlange schnell zunimmt. Sie könnten beispielsweise an Verkehrsschwerpunkten eingesetzt werden, an denen mit relativ glattem Datenverkehr aber hoher Ankunftsrate gerechnet wird. Die Funktionen $f_1(l)$ und $f_3(l)$ wirken stärker auf kurze Bursts als $f_0(l)$. Ihre Warteschlange wird schneller wachsen als die von $f_0(l)$, wobei die von $f_3(l)$ schneller wächst als die von $f_1(l)$. Bei längeren Bursts werden diese Warteschlangen schneller überlaufen als Warteschlangen mit $f_0(l)$ als Verzögerungsfunktion. Dabei ist diese Eigenschaft bei $f_1(l)$ wiederum weniger ausgeprägt als bei $f_3(l)$. Diese Klasse von Verzögerungsfunktionen eignet sich daher für Warteschlangen, die viele kleine Bursts stark verformen sollen und sehr selten längerer Überlast ausgesetzt sind. Solcher Datenverkehr kann zum Beispiel in Zugangsnetzen entstehen.
Mit den gewählten Vertretern dieser Funktionsgruppen wird das Spektrum der in Frage kommenden, einfachen Funktionen relativ gut abgedeckt. Darüber hinaus ist es über den

Umfang dieser Arbeit hinaus möglich, Aussagen über partielle Verzögerungsfunktionen zu treffen, die aus einfachen Funktionen zusammengesetzt sind.

4. Umsetzungsanalyse und Implementierung

4.1. Einführung in ns2 und Hilfswerkzeuge

Der Netzwerksimulator ns2 ist ein System zur Simulation komplexer Vorgänge in Netzwerken auf Basis von Warteschlangen. Er ist in C++ implementiert und wird über ein oTcl-Interface gesteuert und konfiguriert [3]. Er besitzt einen oTcl Interpreter, Simulationsszenarien sind oTcl Skripte, die durch den Simulator interpretiert und ausgeführt werden. Der ns2 ist auf Grund seiner freien Verfügbarkeit, guten Erweiterbarkeit, der Verfügbarkeit des Quelltextes, hohen Flexibilität und relativ guten Dokumentation *das* Werkzeug für Netzwerksimulationen im akademischen Bereich. Aus diesem Grund wurde er als Zielsystem für die Implementierung und Evaluation der Burst Shaping Queue ausgewählt.
Als Ergebnis eines Simulationslaufes kann man verschiedene Ergebnisdateien erstellen, die mit Werkzeugen wie nam (Netzwerk-Animator) oder xgraph (Funktionsplotter) visualisiert werden können.
Umsetzung, Test und Beurteilung der Burst Shaping Queue werden in drei Phasen erfolgen:

- Implementierung der Burst Shaping Queue als oTcl-Klasse in C++
- Test der Implementierung durch geeignete einfache ns2-Szenarien
- Auswertung der gewonnen Testdaten und Bewertung

4.2. Analyse des Warteschlangenmodells des ns2

Im ns2 werden Warteschlangen durch Instanzen von Warteschlangenklassen abgebildet. Um ein Objekt dieser Klassen durch Simulationsskripte zu instanzieren und zu manipulieren muss zusätzlich noch eine an diese C++ Klasse gebundene oTcl-Objekt-Klasse existieren, die die Schnittstellen der C++ Klasse exportiert.
Jede im ns2 nutzbare Queue-Klasse muss von der abstrakten Basisklasse *Queue* abgeleitet werden, die die allgemeine Eigenschaften und Methoden einer Warteschlange kapselt. Jedes Queue-Objekt ist wiederum von der Klasse *Connector* abgeleitet, die Eigenschaften allgemeiner, den Datenfluss weiterleitender Objekte zusammenfasst. Ein Queue-Objekt hat je einen Up- und Downstream-Nachbarn, also Connector-Objekte, die Pakete in die Warteschlange einstellen bzw. bediente Pakete entgegennehmen. Zu

jeder *Queue* gehört eine *QueueHandler* Klasse, die als Eventhandler für die Klasse fungiert und anderen Klassen callback-Funktionen zur Verfügung stellt. Zur Benutzung von C++ Objektinstanzen in oTcl-Skripten muss eine von *TclClass* abgeleitete Kapselklasse zur Anbindung der C++ Klasse an ein oTcl Interface existieren. Diese Kapselklasse ist für die Instanzierung der C++ Klasse verantwortlich, wenn ein Objekt der Kapselklasse in einem oTcl-Skript erzeugt wird. Weiterhin bildet sie die Attribute und Methoden der C++ Klasse geeignet in den oTcl Namensraum ab.
Jedes Paket wird durch ein Objekt der Klasse *Paket* repräsentiert. Das folgende Klassendiagramm visualisiert die eben geschilderten Zusammenhänge ;eine genaue Dokumentation und Quellcode finden sich unter [3].

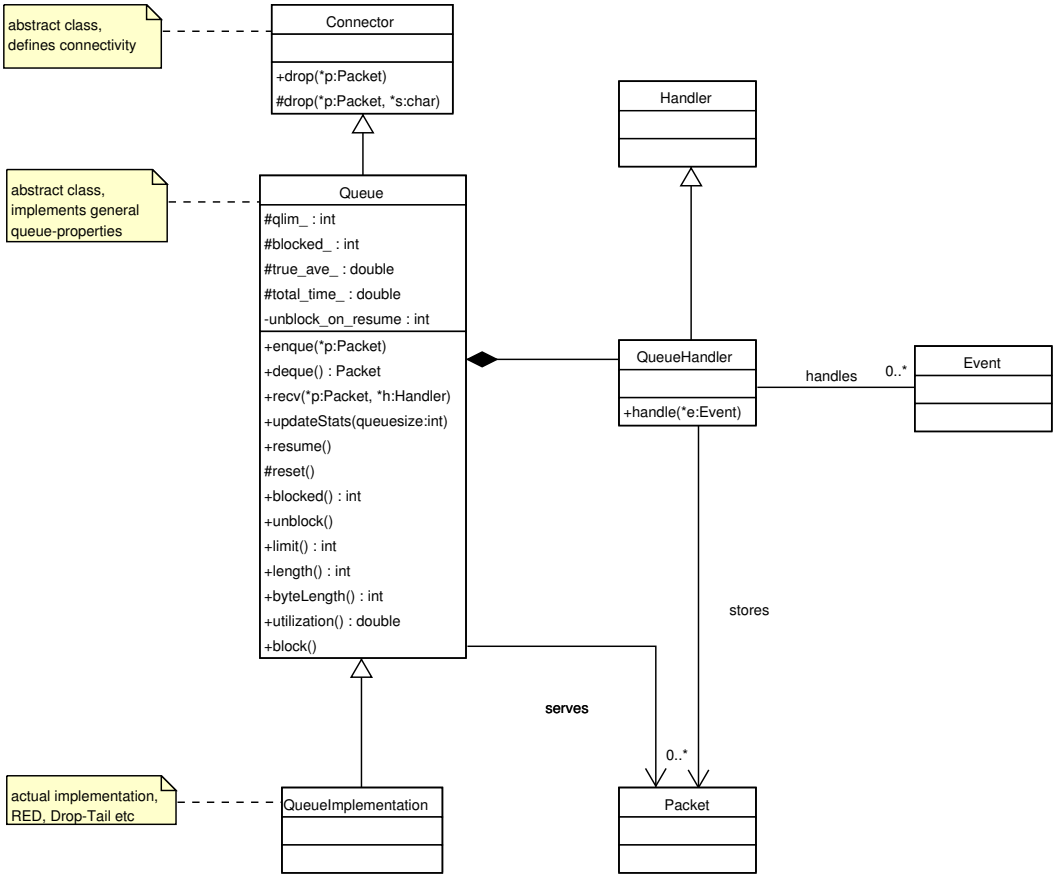

Abbildung 4.1.: Klassendiagramm ns2 QueueImplementierung

Abgesehen vom Zusammenspiel der an der Umsetzung des Warteschlangenkonzeptes beteiligten Klassen ist die innere Struktur der abstrakten Klasse *Queue* von besonderer Bedeutung. In ihr ist das grundsätzliche Verhalten jeder Warteschlange abgebildet. Ein Objekt der Klasse *Queue* kennt zwei Basiszustände: *blockiert* und *nicht blockiert.* Eine Queue ist im Zustand *blockiert,* solange ein Paket bedient wird. Die Zeitdauer, in der die Queue sich in diesem Zustand befindet, wird durch die Bediengeschwindigkeit (d.h. die Linkrate) des Downstream-Nachbarn und die Länge des zu bedienenden Paket-Objektes bestimmt.

Ein Zustandswechsel nach *nicht blockiert* erfolgt, wenn das Paket vollständig bedient ist und ein neues Paket bedient werden könnte. Wenn keine Pakete in der Queue warten, ändert sich ihr Zustand auf *nicht blockiert.* Sollten noch Pakete warten, wird das nächste Paket-Objekt bedient und die Queue verbleibt blockiert. Der Zustandswechsel der Queue wird durch den Aufruf der Methoden *recv(Packet *p, Handler *h)* durch den Upstream-Nachbarn bzw. *resume()* durch den Queue-Handler, initiert durch den Downstream-Nachbarn, ausgelöst. Die Abarbeitung einer Queue erfolgt durch Aufruf der Methoden *enqueue(Packet *p)* und *dequeue()*, die bei Eintreffen bzw. nach Bedienung eines Paketes durch *recv(Packet *p, Handler *h))* bzw. *resume()* aufgerufen werden. Diese Methoden sind in den von *Queue* abgeleiteten Klassen implementiert und beinhalten die eigentliche Umsetzung der Bedienstrategie.
Der Simulationsablauf im ns2 wird durch einen zentralen Scheduler gesteuert, der die Simulationszeit verwaltet und den Eintritt von Ereignissen an einem bestimmten Simulationszeitpunkt ermöglicht. Alle anderen Objekte und damit der komplette Simulationsablauf werden so durch die vom Scheduler erzeugten *Events* zentral gesteuert. Die Generierung von Ereignissen durch den Scheduler kann sowohl durch Objekte als auch durch den Programmierer des Systems direkt durch das Festlegen von konkreten Simulationszeitpunkten und zu benachrichtigender Objekte beeinflusst werden. Der Scheduler ruft dann zu diesem Zeitpunkt die registrierten CallBack-Methoden dieser Simulationsobjekten mit *Event* Objekten als Parametern auf. Dieser Planungsvorgang wird *scheduling* genannt. Sollten zwei dieser Events zum gleichen Simulationszeitpunkt geschedult werden, wird als erstes der zuerst im Scheduler geplante Event ausgelöst. Im Simulationsskript wird der Scheduler zusammen mit anderen Eigenschaften und Methoden in der die Simulation repräsentierenden *simulator*-Klasse gekapselt.

4.3. Modellierung der Burst Shaping Eigenschaft

Die Burst Shaping Queue führt, wie im letzten Kapitel formal beschrieben, für jedes zu bedienende Paket eine Verzögerung t_d ein. Die Umsetzung der Verzögerung ist auf Grund der Eventsteuerung und der Scheduling-Mechanismen des ns 2 sehr gut durchführbar. Dazu müssen die Funktionen *recv(Packet *p, Handler *h)* und *resume()* der jeweiligen Warteschlangenklasse verändert werden. Sie legen den Aufrufablauf von *enqueue(Packet *p)* und *dequeue()* fest und steuern so die eigentliche Implementierung der Bedienstrategie. Die starre, zeitlich unverzögerte Aufrufreihenfolge von *enque(Packet *p)* und *deque()* in der ursprünglichen Implementierung der Klasse *Queue* wird so aufgebrochen. Die Nutzung eines *Timer*-Objektes, das den zeitlichen Abstand zwischen den Funktionsaufrufen regelt, ermöglicht eine Steuerung der Aufrufzeitpunkte und damit eine Umsetzung der Verzögerung.
Um dies zu erreichen, können zum einen diese Methoden direkt in den die Bedienstrategie umsetzenden Klassen, z.B. *DropTail*, implementiert werden. Damit werden die Methoden der Basisklasse *Queue* bei Nutzung von später Bindung (late binding) überschrieben. Das würde eine Neuimplementierung oder Modifikation jeder Bedienstratgie, die mit Burst Shaping Eigenschaften ausgestattet werden soll, zur Folge haben. Der Implementierungsaufwand dafür ist verhältnismäßig hoch. Zum anderen kann auch die Klasse *Queue* modifiziert werden. Der Implementierungsaufwand ist damit nur einmal vorhanden und jede von *Queue* abgeleitete Klasse hätte nun zwangsläufig Burst Shaping Eigenschaften. Die ursprüngliche Implementierung der Klasse *Queue* im ns2 und ihr Verhalten wären so nicht mehr nutzbar. Die dritte Möglichkeit besteht in der Einführung einer abstrakten Basisklasse *QueueBS*, die in der Vererbungshierachie zwischen *Queue* und einer eine Bedienstrategie implementierenden Klasse wie RED oder DropTail steht. Diese Klasse wird von *Queue* abgeleitet und kapselt die Burst Shaping Eigenschaft durch Neuimplementierung der Methoden *recv(Packet *p, Handler *h)* und *resume()*. Von dieser werden dann die eigentlichen Queueimplementierungen abgeleitet und deren Bedienstrategie durch Implementierung der Methoden *enque(Packet *p)* und *deque()* realisiert. Die Eigenschaften der Burst Shaping Queue werden so nicht nur in einer eigenen Basisklasse *QueueBS* gekapselt, sondern die ursprüngliche Vererbungshierachie des ns2 wird durch diese Erweiterung erhalten. Sie kann weiterhin unverändert genutzt werden. Die neue Klasse implementiert dabei nur die zur Umsetzung der Burst Shaping Eigenschaften notwendigen Schnittstellen der Klasse *Queue* neu, der Rest wird durch die Ableitung von *Queue* erhalten. Klassen, die Bedienstrategien implementieren, erhalten so Burst Shaping Eigenschaften allein durch Ableitung von der neuen Basisklasse. Damit bietet sich die Möglichkeit, alle schon vorhandenen Implementierungen von Warteschlangen durch eine einfache Änderung der Vaterklasse mit Burst Shaping-Eigenschaften auszustatten. Dabei können Warteschlangenimplementationen wahlweise, d.h. nur durch Änderung der Vaterklasse, mit oder ohne BurstShaping Eigenschaften ausgestattet werden. Es besteht so nur der Aufwand für die Implementierung der neuen Basisklasse und die Änderungen in den schon bestehenden Implementierungen. Von allen vorestellten Alternativen nutzt die Einführung einer neuen Basisklasse das konsequent objektorientierte Design des ns2 am Besten. Sie bietet eine elegante und effiziente Lösung, da sie übersichtlich, gut strukturiert und leicht wiederzuverwenden ist. Gleichzeitig ist der Implementierungsaufwand relativ gering.

Das nachfolgende Klassendiagramm 4.2 verdeutlicht die neue Vererbungshierachie.

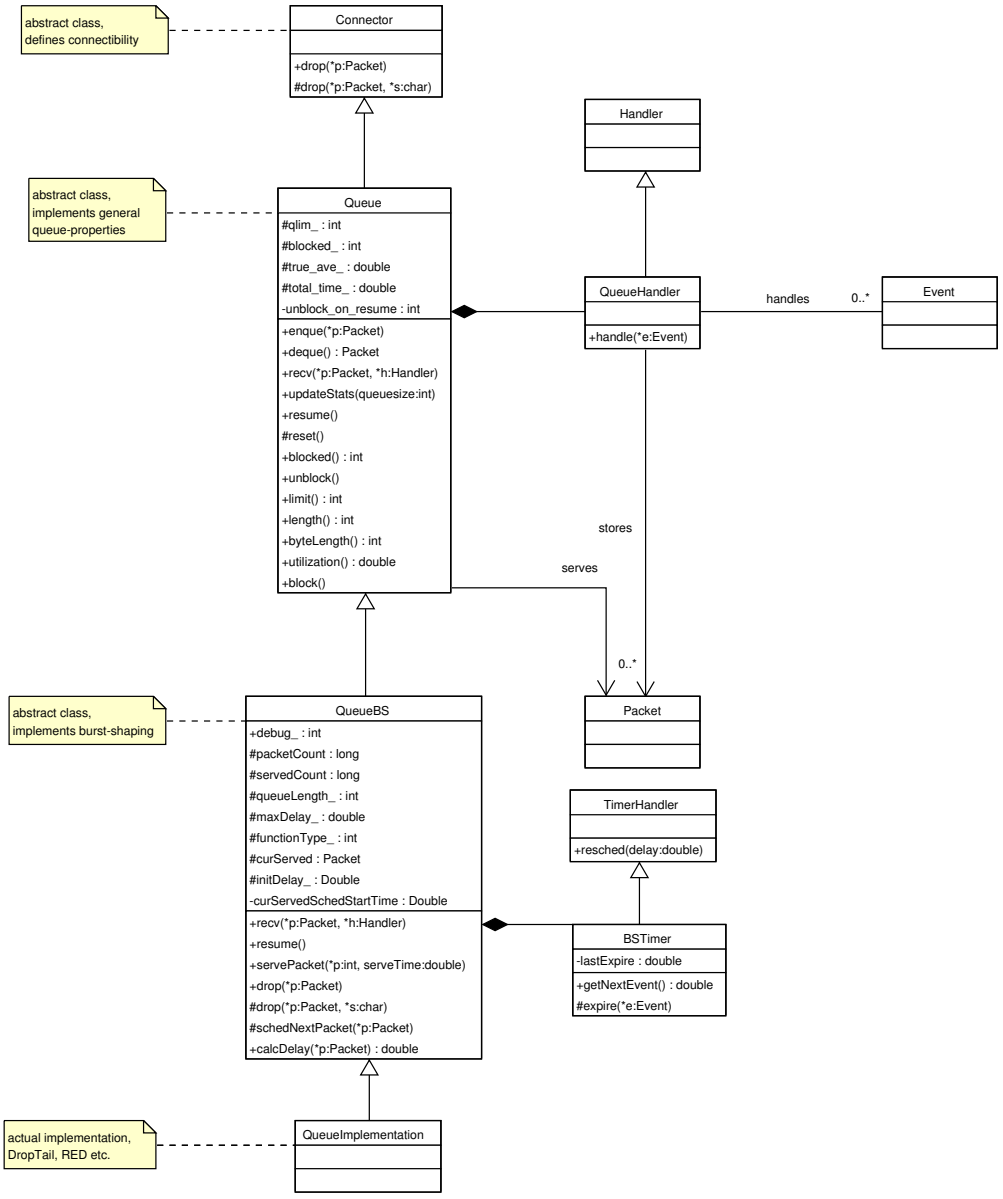

Abbildung 4.2.: Klassendiagramm ns2 QueueBSImplementierung

4.4. Implementierung der C++ Klasse QueueBS

In der Klasse *QueueBS* wird der in Abbildung 3.1 abgebildete erste Algorithmenentwurf objektorientiert umgesetzt. Die neue Klasse *QueueBS* überschreibt die von *Queue* geerbten Methoden *recv(Packet *p, Handler *h)* und *resume()*. Weiterhin führt sie drei neue Methoden ein:

- *void schedNextPacket(Packet p)* – Bedienzeitpunkt für Paket festlegen
- *void servePacket(Packet p)* – Paket bedienen
- *double calcDelay(Packet p)* – Verzögerung für Paket berechnen

Die von *Connector* geerbten *drop* Methoden werden ebenfalls überschrieben, um die Länge der internen Warteschlange völlig unabhängig von der eigentlichen Implementierung zu verwalten. Darüber hinaus werden für die Klasse *QueueBS* Attribute wie folgt definiert:

- *protected long packetCount_* – Zähler für Statistiken
- *protected long servedCount_* – Zähler für Statistiken
- *protected int queueLength_* – aktuelle Queuelänge
- *protected double maxDelay_* – maximale Verzögerung
- *protected double initDelay_* – initiale Verzögerung
- *protected int functionType_* – Flag zum Wählen der Verzögerungsfunktion
- *protected Packet curServed* – aktuell bedientes Paket
- *private double curServedSchedStartTime* – Bedienungsbeginn

Zum exakten Steuern der Ausführung der Methode *servePacket(Packet *p)* in Abhängigkeit von der Simulationszeit erhält die Klasse *QueueBS* ein Timer-Objekt *timer* der Klasse *Scheduler*, die von *TimeHandler* abgeleitet ist. Die Klasse *TimeHandler* implementiert die Methode *expire(Event *e)*, die bei Eintreffen des Ereignisses e, d.h. bei Ablauf des Timers, als callback-Methode für den Scheduler dient und die Methode *servePacket(Packet p)* aufruft [3]. Durch diese Konstruktion wird die verzögerte Bedienung eines Paketes realisiert. Genau wie in der Klasse *Queue* definiert, kennt auch die Klasse *QueueBS* die Zustände *blockiert* und *nicht blockiert*, die die Bedienung eines Paketes beschreiben. Eine *Queue* ist blockiert, wenn gerade ein Paket bedient wird. In Burst Shaping Queues wurde der Begriff effektive Bedienzeit auf Verzögerung und Bedienzeit ausgedehnt. Damit sich eine *QueueBS* nach außen wie eine *Queue* verhält, wird der Status *blockiert* auf die effektive Bedienzeit ausgedehnt. Das bedeutet, dass sich eine *QueueBS* auch im Zustand *blockiert* befindet, wenn ein Paket verzögert wird. Innerhalb der effektiven Bedienzeit eines Paketes, d.h. des Zustandes *blockiert*, ist die Einführung eines weiteren, internen Zustandes nötig, um zwischen der Verzögerung eines Paketes und seiner tatsächlichen Bedienung unterscheiden zu könnnen. Wenn ein

Paket verzögert wird, heißt dieser Zustand *wait*, sein Gegenteil *nowait*. Die Abbildung des Zustandes erfolgt über den Wert der Variablen *curServed*, die die Assoziation zum gerade bedienten *aktuellen Paket* abbildet. Diese Variable ist NULL, wenn kein Paket verzögert wird.
Für die Burst Shaping Queue werden daher folgende Zustände in Anlehnung an die Klasse *Queue* definiert. Die Ursprungszustände der Klasse *Queue* wurden zur eindeutigen Zuordnung für die Klasse *QueueBS* neu benannt.

- **notBlocked** – es befindet sich kein Paket in Bedienung
- **blocked & wait** – Paket wird bedient und verzögert
- **blocked & nowait** – Paket wird „echt“ bedient

Wie das folgende Diagramm 4.3verdeutlicht, kann das Verhalten der Burst Shaping Queue mit Hilfe dieser Zustandsdefinition vollständig beschrieben werden.

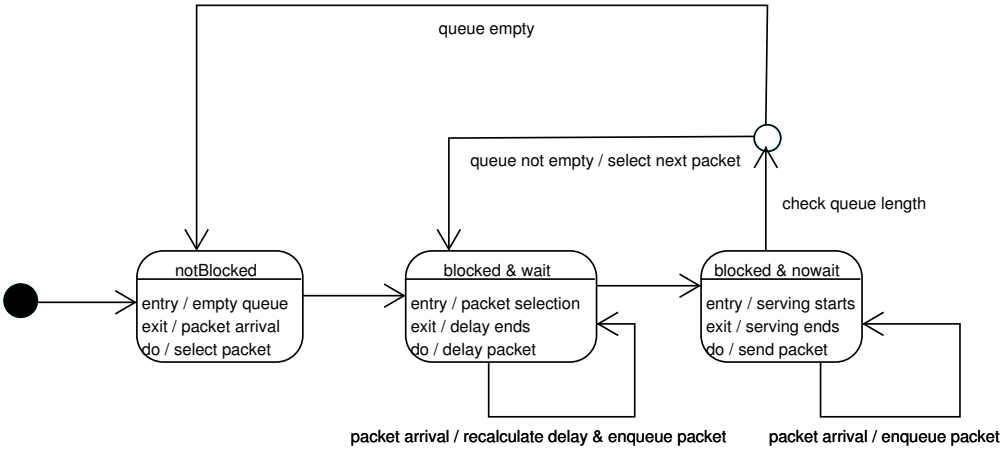

Abbildung 4.3.: Interne Zustände der QueueBS

Eine Burst Shaping Queue mit leerer Warteschlange befindet sich im Zustand *notBlocked*. Das Eintreffen eines Paketes löst dessen Auswahl als aktuelles Paket zur Bedienung aus. Ist diese erfolgreich, wird der Zustand *blocked & wait* eingenommen – das Paket wird verzögert. Das Eintreffen weiterer Pakete während dieses Zustandes führt zur Aufnahme dieser in die Warteschlange und der Neuberechnung der Verzögerung für das aktuelle Paket. Die Warteschlange verbleibt dabei im aktuellen Zustand. Ein Wechsel nach *blocked & nowait* erfolgt erst, wenn die Verzögerung des aktuellen Paketes beendet wird und mit dessen tatsächliche Bedienung begonnen wird. Wenn in diesem Zustand weitere Pakete eintreffen werden sie ebenfalls in die Warteschlange eingestellt. Nach Ende der tatsächlichen Bedienung des Paketes wird die Warteschlange auf weitere zu bedienende Pakete geprüft. Existiert mindestens ein solches Paket, wechselt die Warteschlange in den Zustand *blocked & wait* und bedient diese Pakete. Ist das nicht der Fall, nimmt sie den Ausgangszustand *notBlocked* wieder ein.
Die neu eingeführten Methoden für die Klasse *QueueBS* setzen im Zusammenspiel mit

den internen Zuständen der Klasse das Verhalten der Burst Shaping Queue um. Dabei wird die Neuberechnung der Verzögerung des aktuellen Pakets durch die Methode *schedNextPaket* erreicht. Diese Methode ist Bindeglied zwischen der eigentlichen Neuberechnung der Verzögerung durch *calcDelay* und dem zum *QueueBS*-Objekt gehörigen *BSTimer*-Objekts. Darüber hinaus wandelt diese Methode die berechnete Verzögerung in eine Verzögerung relativ zur aktuellen Simulationszeit um. Dies ist nötig, da sich die Verzögerung für das aktuelle Paket während seiner Wartezeit durch Ankunft weiterer Pakete ändern kann. Die Klasse *BSTimer*, die die Verzögerung steuert, erwartet aber Übergabewerte relativ zur aktuellen Simualtionszeit. Dieses Objekt legt im Scheduler den Zeitpunkt des Aufrufes der eigentlichen Bedienmethode *servePaket* fest, die das aktuelle Paket dann tatsächlich bedient.
Das Eintreffen eines Paketes in einer Burst Shaping Queue wird durch den Aufruf ihrer Methode *recv(Packet *p, Handler *h)* durch das versendende *Container*-Objekt abgebildet. Innerhalb dieser neu implementierten Methode wird als erstes das neu empfangene Paket durch Aufruf von *enque()* in die Warteschlange eingestellt. Danach wird durch Aufruf der Methode *blocked()* überprüft, in welchem Zustand sich die Burst Shaping Queue gerade befindet. Wenn kein Paket in Bedienung ist, wird aus der Queue das nächste mit *deque* ausgewählt und im Attribut *curServed* abgelegt. Anderenfalls ist an *curServed* das sich gerade in Bedienung befindliche Paket assoziiert, dessen Verzögerung nun neu berechnet werden muss. Zur Neuberechnung wird *schedNextPacket(Packet *p)* aufgerufen, das mit *calcDelay* die Verzögerung für das aktuelle Paket neu berechnet. Danach wird die berechnete Verzögerung relativ zur aktuellen Simulationszeit transformiert und der Bedienzeitpunkt des aktuellen Pakets per Aufruf von *resched(Packet *p)* des zur Burst Shaping Queue gehörenden *BSTimer* Objektes festgelegt. Die *resched*-Methode wird genutzt, da sie sowohl einen neuen Timer für ein neu zu bedienendes aktuelles Paket erzeugen, als auch den schon bestehenden Timer für ein aktuelles Paket, dessen Bedienzeitpunkt sich verlagert, ändern kann. Im oben abgebildeten Zustandsdiagramm 4.3 bilden die eben geschilderten Abläufe die Zustandsübergänge **notBlocked** → **blocked & wait** sowie **blocked & wait** → **blocked & wait** und sind im Wesentlichen in der Methode *recv(Packet *p, Handler *h)* implementiert. Das folgende Sequenzdiagramm 4.4stellt diese Zusammenhänge noch einmal dar.

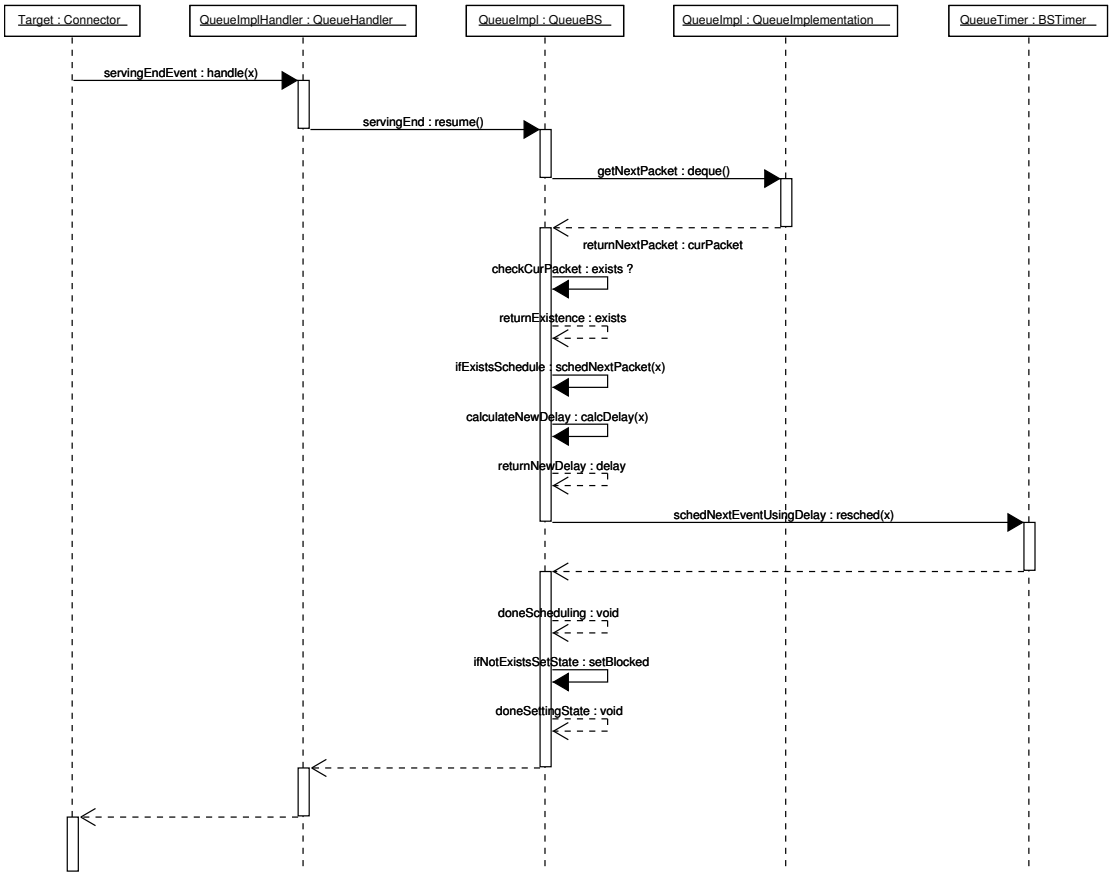

Abbildung 4.4.: Sequenzdiagramm Paketempfang und Verzögerungsneuberechnung

Der Zustandsübergang **blocked & nowait → blocked & nowait** wird ebenfalls von der Methode *recv(Packet *p, Handler *h* implementiert. Er ist sehr einfach: sollte während der „echten Bedienung“ des aktuellen Paketes ein neues Paket ankommen, wird es einfach in die Warteschlange gestellt. Sonst passiert nichts.
Wenn die Simulationszeit das Ende der Verzögerung erreicht, beginnt die „echte“ Bedienung des aktuellen Paketes. Dies geschieht durch das Generieren eines Ereignisses durch den Scheduler und der Übergabe diesen an die CallBack-Methode *expire* des *BSTimer* Objektes. Der genaue Eintritt dieses Ereignisses und damit der Aufruf der CallbackMethode wurde vorher durch *resched* festgelegt. Die Methode *expire* ruft nun die Methode *servePacket* des *QueueBS*-Objektes auf – die eigentliche Bedienung beginnt. Die Bedienung wird durch den Aufruf der *recv*-Methode des assoziierten *Container*-Objektes, das das aktuelle Paket empfängt, ausgelöst. Dieser Zusammenhang stellt den Zustandsübergang *blocked & wait → blocked & nowait* des Zustandsgraphen dar. Das folgende Sequenzdiagramm 4.5 verdeutlicht diese Zusammenhänge.

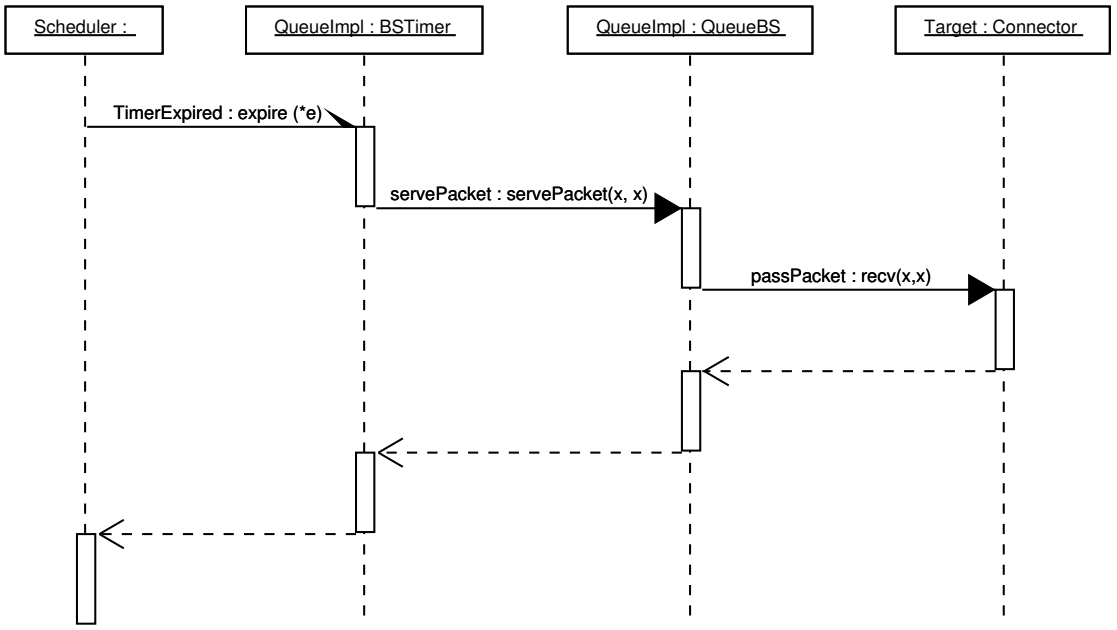

Abbildung 4.5.: Sequenzdiagramm Ankunft eines neuen Paketes bei leerer Warteschlange

Zur Simulation der Bedienzeit des Paketes, also der Übertragungszeit durch das assoziierte *Container* Objekt hindurch zur nächsten Queue, wird das Ende der Bedienung durch einen separaten Methodenaufruf signalisiert. Dieser Methodenaufruf erfolgt durch den zur Burst Shaping Queue gehörigen *QueueHandler*. Er ruft *resume* auf – die Bedienung des aktuellen Paketes ist genau zu diesem Zeitpunkt abgeschlossen. Die Queue überprüft nun per Aufruf von *deque*, ob sich noch Pakete in der Queue befinden. Dies geschieht durch Prüfung des Rückgabewertes, der in *curServed* abgelegt wird. Sollte kein Paket mehr vorhanden sein, d.h. der Rückgabewert ist NULL, wird das Zustandsattribut *blocked* angepasst und die Methode ist beendet. Die Anpassung des Zustandsattributes erfolgt nur zur Wahrung der Kompatiblität zum Verhalten der Klasse *Queue*, das durch das geerbte Attribut *unblock_on_resume* gesteuert werden kann. Dieses Flag regelt den Wert des Attributes *blocked* nach Aufruf der *resume*-Methode.
Wenn sich anderenfalls noch mindestens ein Paket in der Warteschlange befindet, so enthält *curServed* nun einen Pointer auf das neue aktuelle Paket, das von *deque()* zurückgeliefert wurde. Für dieses Paket wird nun analog zum Sequenzdiagramm in Abbildung 4.4 die Verzögerung berechnet und der Bedienzeitpunkt festgelegt. Dazu wird *schedNextPacket(Packet *p)* aufgerufen und damit ein Aufruf von *calcDelay(Packet *p)* und der Methode *resched(double delay)* des zur *QueueBS* gehörenden *TimerHandlers*. Dieser Ablauf bildet schließlich die letzten beiden Zustandsübergänge *locked & nowait* → *notBlocked* und *locked & nowait* → *blocked & nowait* ab und ist im nachfolgenden Sequenzdiagramm in Abbildung 4.6 dargestellt.

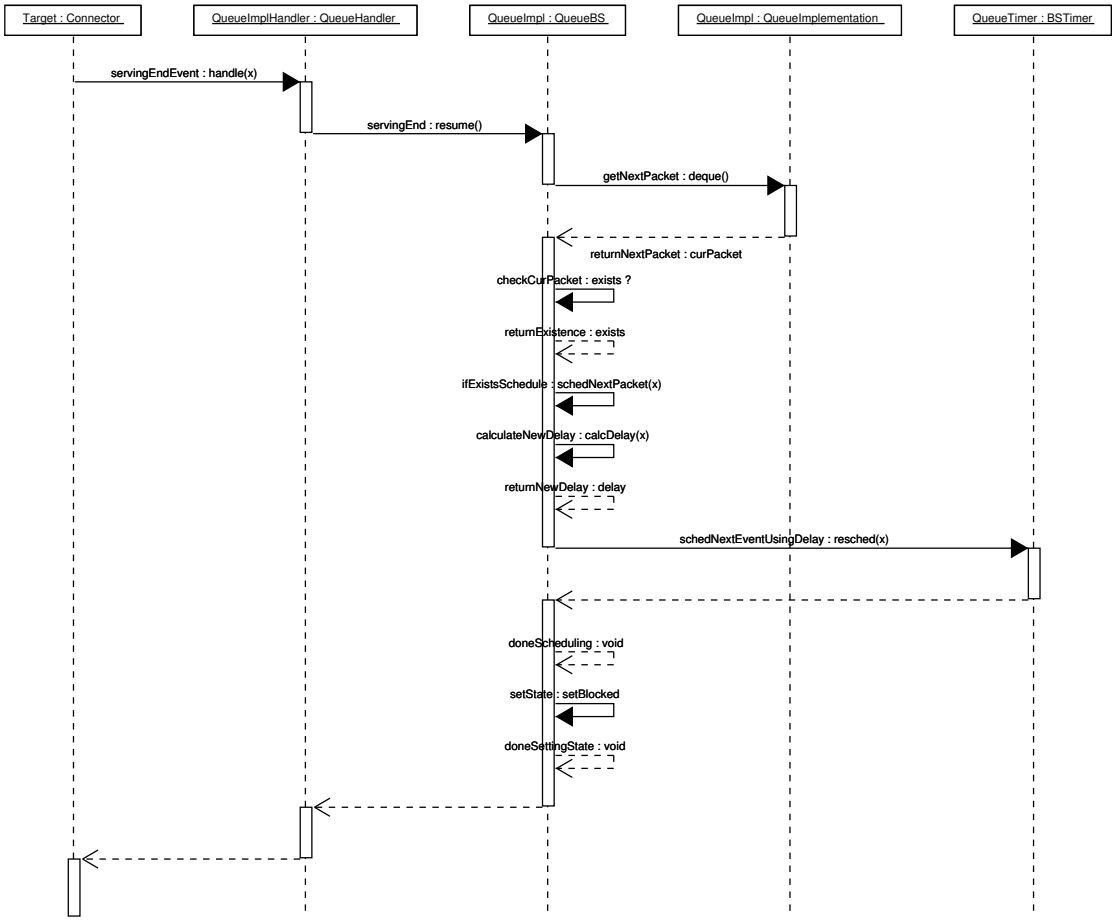

Abbildung 4.6.: Sequenzdiagramm Bedienung zweier aufeinanderfolgender Pakete

Wie schon erwähnt, steuert das Flag *unblock_on_resume* der Klasse *QueueBS*, das von *Queue* geerbt wurde, den Wert des *blocked* Attributes nach Beendigung der *resume*-Methode. Ist der Wert des Attributes „true“, dann wird sich die Burst Shaping Queue wie eine normale *Queue* verhalten. Dabei werden alle Pakete in die Warteschlange gestellt und kein einziges bedient. Ein Objekt der Klasse *QueueBS* wird sich so in einem Zustand befinden, der nicht im Zustandsgraph aufgeführt ist. Es wurde nicht im Zustandsgraph abgebildet, da dieser Zustandsübergang nur von „außen“, d.h. durch das Simulationsskript herbeigeführt werden kann und kein beabsichtigtes Verhalten der Klasse *QueueBS* darstellt.

4.5. Integration und Nutzung der C++ Klasse QueueBS in ns2

Dieser Abschnitt erläutert kurz die wesentlichen Schritte, um die in Abschnitt 4.4 vorgestellte Implementierung der BurstShaping Queue im ns2 nutzen zu können. Dieses Vorgehen hat keinen Einfluss auf die eigentliche Umsetzung des Algorithmus sondern ist durch die interne Struktur des ns2 bedingt. Die im Folgenden geschilderten Schritte können mit einem korrekt installierten und kompilierten ns2, Version 2.26, nachvollzogen werden. Dabei wird die Verwendung des Paketes ns-allinone-2.26.tar.gz empfohlen, das neben dem ns2 auch oTcl, nam und weiter notwendige Bibliotheken und Werkzeuge enthält. Dieses Paket ist unter [1] erhältlich.
In Simulationen ist es notwendig, Parameter einer Burst Shaping Queue, wie minimale Verzögerung oder Schwellwert, festzulegen oder zu verändern. Die Simulationen werden durch die Abarbeitung von oTcl-Skripten gesteuert. Die Manipulation von Warteschlangenparametern erfordert daher einen gemeinsamen Zugriff der C++ Implementierung und des oTcl-Skripts auf die Adressräume dieser Parameter. Der einfachste Weg dies zu erreichen, ist die Bindung von C++ Attributen an oTcl Variable durch Aufruf der Funktion *bind* für jedes zu exportierendes Attribut [3]. Dieser Aufruf erfolgt im Konstruktor der Klasse *QueueBS* und ermöglicht so eine Veränderung der Variablen eines Objektes dieser Klasse in oTcl Skripten, sobald eine Instanz von ihm erzeugt wurde. Darüber hinaus ist es sinnvoll, die Attribute mit Default-Werten zu belegen. Das kann zum einen direkt im C++ Quelltext, zum anderen durch oTcl geschehen. Eine Festlegung im Quelltext würde bei jeder Änderung eine erneute Übersetzung des Quelltextes erfordern. Zum anderen muss ein Nutzer, der die Default-Werte ändern will den Quelltext ebenfalls ändern. Im Gegensatz dazu ist eine Festlegung im oTcl System eleganter, da es eine zentrale, standardisierte Datei gibt, in der alle Default-Werte für alle oTcl Klassen abgelegt werden. Eine Änderung dieser Datei erfordert keine Neukompilierung zur Übernahme der Werte. Daher wird diese Lösung bevorzugt. Im Verzeichnis des ns2 befindet sich die Datei *tcl/lib/ns-default.tcl*, die die Default-Werte für die meisten gebundenen Variablen enthält [3]. Dort werden neue Einträge für die neue oTcl-Klasse erstellt und die gebundenen Werte belegt.
Ein anderer Weg, dieses Ziel zu erreichen, ist die Erzeugung von oTcl Accessormethoden für die zu exportierenden Attribute der C++ Klasse. Diese Methode entspricht vielmehr der in der Objektorientierten Softwareentwicklung geforderte Datenkapselung als das einfache Exportieren der Attribute. Darüber hinaus bietet sie die Möglichkeit, übergebene Werte auf Validität zu prüfen. Accessormethoden verursacht aber einen hohen Implementierungsaufwand, der im Kontext dieser Arbeit nicht gerechtfertigt ist. Aus diesem Grund werden die notwendigen Attribute der Klasse *QueueBS* durch Bindung an oTcl Variable verfügbar gemacht. Accessormethoden stellen eine mögliche Erweiterung dar, die insbesondere im Zuge einer eventuellen Nutzung der Implementierung durch Dritte unbedingt vorgenommen werden sollte.
Damit in oTcl Skripten ein Objekt, das durch eine C++ Klasse implementiert worden ist, instanziert werden kann, muss es an eine oTcl Klasse gebunden werden. Da die Implementierung der Burst Shaping Eigenschaft einer Warteschlange in einer abstrakten C++ Klasse erfolgt, ist sie nicht direkt instanzierbar. Sie kann daher auch nicht an eine

oTcl Klasse gebunden werden. Es ist dadurch unmöglich, die neue Veerbungshierachie in den oTcl Namensraum abzubilden. Vielmehr ist es nötig, die C++ Klasse, die die eigentliche Bedienstrategie implementiert und von *QueueBS* erbt, an eine eigene oTcl Klasse zu binden. Über die Instanzvariablen dieses oTcl Objektes kann dann auf die exportierten Attribute der Klasse *QueueBS* zugegriffen werden. Für die Simulationen im folgenden Kapitel wurde eine Klasse *DropTailBS* erstellt. Sie entspricht der im ns2 schon vorhandenen Implementierung der Klasse *DropTail*, deren Vaterklasse aber von *Queue* auf *QueueBS* geändert wurde. An sie ist die oTcl-Klasse *DropTailBS* gebunden, über deren Instanzvariable eine Einstellung der Parameter der Burst Shaping Eigenschaft möglich wird. In der oTcl Vererbungshierachie ist *DropTailBS* eine Kindklasse von *Queue*. Nach der Bindung der C++ Implementierung an Klassen und Variablen im oTcl Namensraum muss der ns2 schließlich neu kompiliert werden. Dazu müssen die Dateien der Implementierung korrekt abgelegt werden und Anpassungen in der Datei *makefile* des ns2 vorgenommen werden. Die Implementierung umfasst die Dateien:

- *queue.cc*, *queue.h* – ermöglicht Überschreiben von *resume()*
- *queuebs.cc*, *queuebs.h* – Klassen *QueueBS* und *BSTimer*
- *dtBS-queue.cc*, *dtBS-queue.h* – Drop Tail Queue mit Burst Shaping Eigenschaft

Diese Dateien müssen im Verzeichnis *queue* unter dem Verzeichnis der ns2 Installation abgelegt werden. Dort befinden sich die Quelltexte aller Queueimplementierungen. Danach muss die Erstellung der Objektdateien der Implementierung im *makefile* eingetragen werden. Die Liste der zu erstellenden Objekt-Dateien ist der Wert der Variablen OBJ_CC und muss um die Einträge

- *queue/queue-BS.o*
- *queue/dtBS-queue.o*

erweitert werden. Beim nächsten Kompilieren des ns2 wird die Implementierung dann in den Simulationsskripten nutzbar. Konkrete Beispiele zur Benutzung des Burst Shaping Ansatzes in Simulationsskripten befinden sich in den Simulationsskripten zu Kapitel 5 und liegen der Arbeit bei.

4.6. Bewertung der Implementierung

Im letzen Abschnitt dieses Kapitels wird die Implementierung der Burst Shaping Queue kritisch betrachtet und eingeordnet. Im Abschnitt 3.2 und 3.3 wurde das Konzept der Burst Shaping Queue ausführlich diskutiert und eingeordnet. Bei der Betrachtung der wesentlichen Eigenschaft des Burst Shaping Ansatzes, der Regulierung der Bediengeschwindigkeit in Abhängigkeit von der Warteschlangenlänge, wurde das Ma? für die Warteschlangenlänge bewusst vernachlässigt. Dieser Punkt ist für die Diskussion der grundlegenden Eigenschaften des Konzeptes nicht von fundamentaler Bedeutung. Die Abschnitte 3.3 und 3.4 lassen daher auch eine Diskussion des Maßes bewusst aus. Bei der konkreten Implementierung und der daraus resultierenden Verwendbarkeit in der Praxis wird dieses Maß aber relevant.
Ein Messen der Warteschlangenlänge kann im ns2 auf zwei verschiedene Arten erfolgen. Zum einen kann die Anzahl der Pakete bestimmt werden, zum anderen die Länge der Warteschlange in Bytes. In 3.1 wurde die allgemeine Formel

$$Bedienrate = \frac{BedienteDatenmenge}{Zeiteinheit+Verzoegerung} = \frac{BedienteDatenmenge}{effektiveBedienzeit}$$

zur Berechnung der Bedienrate aufgestellt. Die Bedienrate der Warteschlange soll eindeutig einer bestimmten Warteschlangenlänge zugeordnet sein. Auf Grund der konstanten Linkrate kann sie nicht direkt beeinflusst werden. Daher wurde die Verzögerung eingeführt um die Bedienrate regeln zu können. Für eine bestimmte bediente Datenmenge wird die effektive Bedienzeit beeinflusst und damit auch die effektive Bedienrate.
Auf Grund der Paketorientierung der untersuchten Netzwerke sind die bediente Datenmenge und die zur tatsächlichen Bedienung verbrauchten Zeiteinheiten vom aktuellen Paket abhängig. Seine Länge in Bytes wird der bestimmende Faktor, da die Linkrate in Bytes/s angegeben wird. Damit die Eindeutigkeit der Zuordnung der Bedienrate zur Warteschlangenlänge gewahrt bleibt, muss auch sie in Bytes gemessen werden.
Wenn als zusätzliche Bedingung eine konstante, bekannte Paketgröße vorausgesetzt wird, kann aus der Anzahl der Pakete in der Warteschlange direkt auf ihr Länge in Bytes geschlossen werden. Damit reicht die Anzahl der beinhalteten Pakete als Maß aus.
Die im Rahmen dieser Arbeit entstandene Implementierung soll helfen, das grundsätzliche Verhalten des Burst Shaping Ansatzes zu studieren und eine Basis für eventuelle zukünftige Forschungen bieten. Für dieses Ziel sind Benutzerfreundlichkeit und Flexibilität der Implementierung von untergeordneter Bedeutung. Im Vordergrund standen eine fehlerfreie, einfach nachvollziehbare Umsetzung und gute Erweiterbarkeit. Die Implementierung hat den Charakter eines Laborprototypen – sie zeigt die Stärken und Schwächen eines technischen Konzeptes, ist aber keine ausgereifte, zur Benutzung durch Dritte bestimmte Software. Aus diesem Grund unterliegt die Implementierung folgenden, wesentlichen Einschränkungen:

- Die korrekte Funktion des Warteschlangenalgorithmus ist auf konstante Paketgrößen beschränkt, da die Warteschlangenlänge durch Bestimmung der Anzahl wartender Pakete gemessen wird.

- Die Instanzvariablen der Burst Shaping Queue, die durch ein oTcl Objekt gebunden sind, werden nicht auf Fehleingaben geprüft.

Die Klasse *Queue*, von der die Umsetzung der Burst Shaping Eigenschaft abgeleitet ist, bietet schon eine Unterstützung unterschiedlicher Maße für die Warteschlangenlänge. Damit ist es sehr leicht möglich, die Implementierung zu erweitern. Die zweite wesentliche Einschränkung kann durch den Ersatz der einfachen Variablenbindung durch Accessormethoden in den oTcl-Klassen erreicht werden. Da dieses Vorgehen in [3] gut dokumentiert ist, sollte diese Erweiterung leicht vorzunehmen sein. Mit diesen Eigenschaften sowie entsprechend angepasster Dokumentation kann die Implementierung zur Verwendung durch Dritte freigegeben werden.

5. Testszenarien und Testergebnisse

Im Kapitel 3 wurde der Burst Shaping Ansatz theoretisch analysiert und das erwartete Verhalten einer Burst Shaping Queue, also der Kombination der Burst Shaping Eigenschaft mit einer Drop Tail Queue, beschrieben. Darüber hinaus wurden Aussagen über die Auswirkungen verschiedener Verzögerungsfunktionen auf den Eingangsdatenstrom aufgestellt. Die Tests in diesem Kapitel sollen diese Analysen überprüfen.

5.1. Testszenarien und Aufbau der Testskripte

Dieser Abschnitt erläutert die allgemeine Testziele, die Topologie der Tests und den Aufbau der Simulationsskripte.
Im ersten Testszenario wird das grundsätzliche Verhalten einer Burst Shaping Queue im Vergleich zu einer Drop Tail Warteschlange getestet. Das zweite Testszenario überprüft den Einfluss der vorgeschlagenen Verzögerungsfunktionen auf unterschiedlich lange Bursts im Eingangsdatenstrom.
Beide Testszenarien werden in der gleichen, einfachen Testtopologie durchgeführt, unterscheiden sich aber durch den generierten Datenstrom und die verwendete Warteschlange. Die Topologie besteht aus drei Knoten. Der erste Knoten dient als Datenquelle, der Letzte als Senke. Auf dem Weg von Quelle zu Senke befindet sich ein dritter Knoten. Seine Ausgangswarteschlange zur Datensenke ist die zu vermessende Warteschlange. Als simuliertes Übertragungsprotokoll wird UDP verwendet. Da UDP ein verbindungsloses Protokoll ist, werden keine Antwortpakete erzeugt. Daher existiert nur ein Datenfluss von Quelle zu Senke. Als Verkehrsquelle dient ein Constant Bit Rate Verkehrsgenerator (CBR). Ist dieser aktiviert, erzeugt er Pakete einheitlicher Grösse mit einer einstellbaren, konstanten Datenrate. Durch das zeitgesteuerte Aktivieren und Deaktivieren des Verkehrsgenerators in Kombination mit einer genügend hohen Datenrate können Bursts simuliert werden. Mit dieser Konfiguration kann bei jedem Simulationsdurchlauf exakt der gleiche Eingangsdatenstrom für die zu beobachtende Warteschlange am mittleren Knoten erzeugt werden. Der Quellknoten und der Vermittlungsknoten sind mit einer Linkrate von 500 KB/s verbunden, der Vermittlungsknoten und der Knoten mit der Datensenke mit 250 KBit/s. Durch diese Wahl ist es möglich, Überlast und tatsächliche Überlast zu generieren. Beide Verbindungen arbeiten mit einer Verzögerung von 10ms. Alle zu messenden Warteschlangen haben eine Kapazität von 50 Paketen und einen identischen Schwellwert von 35 Paketen. Um die verschiedenen Warteschlangen zu vergleichen, wird das Simulationsszenario für jede zu beobachtende Warteschlan-

ge einmal durchlaufen. Die Länge der Simulation und die verwendete Warteschlange werden über Parameter des Simulationsskriptes gesteuert. Die gewonnenen Messdaten werden dann in gemeinsame Diagramme für jedes Szenario projiziert, um die Warteschlangen vergleichen und bewerten zu können.
Jede Simulation im ns2 wird durch ein oTcl-Skript gesteuert. Die Simulationsskripte für den ns2 besteht im wesentlichen aus zwei Teilen: einer Beschreibung der Topologie und der Steuerung des Simulationsverlaufes. Als Ergebnis eines Simulationsverlaufes können Dateien mit Mess- und Verlaufsdaten erstellt werden. Diese Dateien können dann mit anderen Programmen wie dem Netzwerkanimator nam, Plottprogrammen wie xgraph oder gnuplot bzw. eigenen Auswertungsprogrammen verarbeitet werden.
Die Skripte für die Testszenarien der Burst Shaping Queue bilden die oben beschriebene Topologie ab, die die folgende Skizze in Abbildung 5.1noch einmal visualisiert. Die

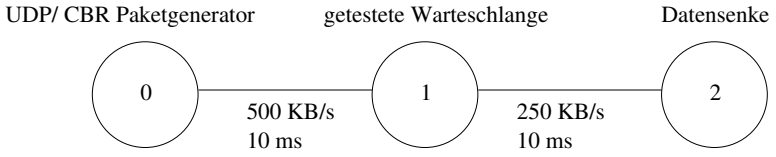

Abbildung 5.1.: Topologie der Testszenarien

in den Testszenarien gewonnenen Messdaten werden mit Hilfe eines Monitorobjektes für die zu beobachtende Warteschlange und der Messprozedur *trace* gewonnen und in Dateien abgelegt. Die Namen der Messdatendatien folgen dem Namensschema:

[Messgroesse]trace[Funktionsindex].out

Der Simulationsverlauf kann bei Aufruf der Simulationsskripte mit drei Parametern gesteuert werden:

- **Typ der Verzögerungsfunktion**
- **Schwellwert der Warteschlange in Paketen**
- **Testdauer in Sekunden**

So startet zum Beispiel *„ns test2.tcl 0 40 16.0“* Testszenario 2a mit Verzögerungsfunktion $f_0(l)$, einem Schwellwert von 40 Paketen und einer Dauer von 16,0 Sekunden. Da die einzelnen Simulationsläufe für jede Warteschlange wiederholt werden müssen und die gewonnen Daten noch durch ein Plotprogramm verarbeitet werden, bietet sich die Möglichkeit die Testläufe komplett zu automatisieren. Daher wurden für jedes Testszenario Unix-Shellskripte erstellt, die den entsprechenden Test durchführen, die Skriptvorlagen für *gnuplot* aus der Vorlagendatei *plot.tmpl* erstellen und schließlich mit *gnuplot* Diagramme erstellen. Shell- und Simulationsskripte sind folgendermaßen den Testszenarien zugeordnet:

- Testszenario 1 – test1a.sh,test1a.tcl
- Testszenario 1 (modifiziert) – test1b.sh, test1b.tcl

- Testszenario 2a – test2.sh, test2.tcl
- Testszenario 2b – test3.sh, test3.tcl
- Testszenario 2c – test4.sh, test4.tcl

Alle Simulations- und Shellskripte befinden sich auf dem der Arbeit beiliegenden Datenträger. Die Diagramme der Messdaten der Testszenarien befinden sich im Anhang.

5.2. Testszenario 1: Allgemeiner Funktionstest

In diesem Abschnitt werden das erste Testszenario und dessen Ergebnisse vorgestellt. Die Ergebnisse werden im Vergleich mit den allgemeinen Analyseerwartungen aus Abschnitt 3.2 diskutiert. Das erste Testszenario testet die Reaktion der Burst Shaping Queue auf verschiedene Ankunftsraten. Dazu wird der Simulationsverlauf in vier aufeinander folgende Phasen geteilt. Der Verkehrsgenerator generiert während der gesamten Simulationszeit einen Datenstrom, dessen Geschwindigkeit zu Beginn jeder Phase geändert wird und danach konstant bleibt. Die zu beobachtende Warteschlange wird mit folgenden Parametern ausgestattet:

- Warteschlangenlänge: 50 Pakete
- Warteschlangenschwellwert: 35 Pakete
- initiale Verzögerung: 0.002 Sekunden
- maximale Verzögerung: 0.01 Sekunden

Die Grösse der generierten Pakete beträgt 500 Bytes. Durch die Wahl der initialen Verzögerung wird die Ankunftsrate, ab der die Burst Shaping Queue den Datenstrom verändert, festgelegt. Sie beträgt hier 125 kBit/s. Der Test wird mit einer Burst Shaping Queue und einer Drop TailQueue mit identischer Warteschlangenlänge als Referenzwarteschlange durchgeführt. Die Simulationsdauer beträgt 14.0 Sekunden, dabei werden 12.0 Sekunden lang Daten generiert. Jede der aufeinander folgenden Phasen dauert 3.0 Sekunden. Auf Grund der Überlegungen in Kapitel 3 werden folgende Vermutungen über den Simulationsverlauf aufgestellt:
In der ersten Phase wird die Warteschlange mit 100 kBit/s, also weniger als der initialen Bedienrate von 125 kBit/s, bedient. Die Warteschlange sollte daher nicht wachsen und die Burst Shaping Queue sollte sich in dieser Phase wie eine Drop Tail Queue verhalten, da keine Überlast herrscht.
In der zweiten Phase wird die Ankunftsrate auf 200 kBit/s erhöht und erzeugt so Überlast in der Warteschlange. Da die maximale Bedienrate der Warteschlange 250 kBit/s (die Linkrate zur Senke) beträgt, entsteht keine tatsächliche Überlast. Die Bedienrate der Warteschlange sollte in dem Moment, in dem die Warteschlange zu wachsen anfängt, kurz auf den durch die maximale Verzögerung festgelegten Wert verringert werden. Danach sollte sie sich sich mit wachsender Warteschlangenlänge erhöhen, bis ein Gleichgewicht zwischen Ankunfts- und Bedienrate hergestellt ist. Die Anpassung der Bedienrate sollte deutlich langsamer als die einer Drop Tail Warteschlange erfolgen.
In der dritten Phase wird die Ankunftsrate wieder auf den ursprünglichen Wert von 100 kBit/s gesenkt. Die Burst Shaping Queue ist wieder unter Normallast und sollte ihre Bediengeschwindigkeit langsamer als eine Drop Tail Queue an die neue Ankunftsrate anpassen. Wenn dieses Gleichgewicht hergestellt ist, sollte sie sich wieder wie eine Drop Tail Warteschlange verhalten.
In der vierten und letzten Phase wird schließlich tatsächliche Überlast durch eine Ankunftsrate von 500 kBit/s erzeugt. Die Burst Shaping Queue sollte bis zum Erreichen des Schwellwertes ihre Bediengeschwindigkeit steigern und sich dann wie eine Drop Tail Warteschlange verhalten. Bei Überschreitung der Warteschlangenkapazität werden

Pakete verworfen. Am Ende dieser Phase wird die Ankunftsrate auf Null gesenkt. Die Burst Shaping Queue sollte ihre Bediengeschwindigkeit ebenfalls auf Null senken. Dabei ist die Abnahme der Warteschlangenlänge langsamer als die der Drop Tail Warteschlange.

Die Ergebnisse der Simulation werden in den Diagrammen im Anhang A dargestellt. Für den Simulationsverlauf wird die Entwicklung folgender Messgrößen für die Drop Tail- und Burst Shaping Queue über den Verlauf der Simulationszeit abgetragen:

- Abbildung A.1 Bedienrate, Ankunftsgeschwindigkeit
- Abbildung A.2 Warteschlangenlänge
- Abbildung A.5 Differenz aus Ankunfts- und Bedienrate
- Abbildung A.3 Datenrate, Ankunftsrate
- Abbildung A.4 Abgelehnte Pakete

Die Graphen der Messdaten bestätigen das erwartete Verhalten der Burst Shaping Queue im Testszenario 1 deutlich, offenbaren aber auch eine schwerwiegende Schwäche der Implementierung.
In der ersten Phase verhält sich die Burst Shaping Queue genau wie eine Drop Tail Queue – sie passt ihre Bedienrate der Ankunftsrate an. Die Graphen in allen Messdiagrammen überdecken sich. In der zweiten Phase passt die Drop Tail- Queue ihre Bedienrate sofort der gestiegenen Ankunftsrate an – beide Graphen überdecken sich wieder. Dagegen gerät die Burst Shaping Queue in eine Überlastsituation und reagiert wie erwartet. Sobald ihre Warteschlange zu wachsen beginnt, vermindert sie die Bediengeschwindigkeit. Daraufhin wächst ihre Warteschlange und sie regelt ihre Bedienrate langsam bis auf das Niveau der Ankunftsrate und behält diese dann bei. Ihre Warteschlange wächst während der Regulierungsphase und bleibt dann auf diesem Niveau konstant. Der Graph der Differenz von Ankunfts- und Bedienrate in Abbildung A.5 zeigt das Regulieren der Bediengeschwindigkeit durch eine Spitze im Graphenverlauf in dieser Phase ebenfalls deutlich. Der Füllstand der Warteschlange verursacht ein zusätzliches Delay der bedienten Pakete – erkennbar am Abstand der Datenratengraphen im Diagramm A.3. Dieses Verhalten bestätigt die Vermutungen über eine Beeinflussung des Delays in Abschnitt 3.2. In der dritten Phase passt die Burst Shaping Queue dann ebenfalls wie die Drop Tail Queue ihre Bedienrate der neuen Ankunftsrate an, wie in den Abbildungen A.1 und A.5 deutlich sichtbar ist. Die Anpassung verläuft erwartungsgemäß deutlich langsamer als bei der Drop Tail Queue. Die Graphen der Warteschlangenlänge in Abbildung A.2 und der Differenz von Ankunfts- und Bedienrate A.5 dagegen offenbaren die schwerwiegende Schwäche der Implementierung in dieser Phase – die Burst Shaping Queue leert ihre Warteschlange nicht vollständig. Dieses Verhalten ist in der dritten Phase, in der Normallast herrscht, nicht erwünscht. Hier soll sich die Burst Shaping Queue wie eine Drop Tail Warteschlange verhalten. Für den weiteren Simulationsverlauf wird nun jedes ankommende Paket auf Grund der partiell gefüllten Warteschlange mit einem zusätzlichen Delay bedient, wie in Abbildung A.3 ersichtlich ist. Eine weitere, viel nachteiligere Konsequenz ist aber erst in der vierten Phase,

in der tatsächliche Überlast herrscht, zu beobachten. Die Bedien- und Datenrate der Burst Shaping Queue verhalten sich, abgesehen vom zusätzlichen Delay und der daraus resultieren Verschiebung des Graphen auf Grund der partiell gefüllten Warteschlange, erwartungsgemäß wie die der Drop Tail Queue. Die Graphen der Warteschlangenlängen in Abbildung A.2 und der abgelehnten Pakete in A.4 zeigen aber die wirklich nachteilige Auswirkung: die Warteschlange läuft auf Grund der schon enthaltenen Pakete deutlich zeitiger über. Als Folge daraus werden viel mehr Pakete durch die Burst Shaping Queue verworfen, als dies bei einer leeren Warteschlange am Ende von Phase drei der Fall wäre.
Die Warteschlange der Burst Shaping Queue wächst – konzeptionell bedingt – schneller als die der Drop Tail Queue und läuft daher etwas eher über. Sie verhält sich damit bei längerer tatsächlicher Überlast geringfügig schlechter. Zusätzliche Pakete in der Warteschlange verringern die verfügbare Kapazität zur Aufnahme von Paketen. Deshalb läuft die Burst Shaping Queue bei einer partiell gefüllten Warteschlange noch zeitiger über und verwirft damit deutlich mehr Pakete als eine Drop Tail Queue. Die Konsequenzen für die Nutzbarkeit des Konzeptes sind fatal.
Bei längerer tatsächlicher Überlast an einer einzelnen Warteschlange sind Paketverluste durch Überlastung der Warteschlange grundsätzlich nicht zu vermeiden. Daher ist das geringfügig schlechtere Verhalten der Burst Shaping Queue im Vergleich zu einer Drop Tail Queue akzeptabel, wenn sie als Ausgleich in anderen Situationen das Überlastverhalten des gesamten Netzwerkes bzw. des jeweiligen Übertragungspfades entscheidend verbessert. Durch unnötigerweise partiell gefüllte Warteschlangen steht in jedem Fall weniger als die maximale Warteschlangenkapazität zur Aufnahme nicht sofort bedienbarer Pakete zur Verfügung. Im schlimmsten Fall ist dies bei tatsächlicher Überlast nur noch die Differenz zwischen Schwellwert und Gesamtkapazität. Damit werden die beabsichtigten Glättungs- und Schutzeffekte des Konzeptes, die ja auf effizienterer Warteschlangennutzung beruhen, eingeschränkt bzw. völlig außer Kraft gesetzt. Der Nutzen der Burst Shaping Eigenschaft ist somit ernsthaft in Frage gestellt.
Die Ursache dieses fehlerhaften Verhaltens ist nicht in der Implementierung zu suchen, sondern grundsätzlicher Natur. Das Verbleiben von Paketen in der Warteschlange in dieser speziellen Situation ist durch in der Wahl von initialer und maximaler Verzögerung zu erklären. Weitere Analyse des Verhaltens führt schließlich zu einer Unzulänglichkeit im zu Grunde liegenden Algorithmus der Burst Shaping Eigenschaft.
In der dritten Phase wird eine Ankunftsrate erzeugt, die zwar kleiner als die initiale Bedienrate ist, aber größer als die Minimale. Die aktuelle Bedienrate aus Phase zwei wird unterschritten, die Warteschlangenlänge fällt und die aktuelle Bedienrate wird gesenkt. Dieser Prozeß wiederholt sich bis zum Erreichen der Ankunftsrate. In diesem Gleichgewichtszustand zwischen Ankunfts- und aktueller Bedienrate kommen genau so viele Pakete in der Warteschlange an, wie sie gleichzeitig wieder verlassen – die angenommenen Warteschlangenlänge wird nicht weiter abgesenkt. Das würde erst bei einer Verminderung der Ankunftsrate geschehen. Eine leere Warteschlange entsteht so nur, wenn die Ankunftsrate unter die minimale Bediengeschwindigkeit fällt, da diese die aktuelle Bedienrate nach unten begrenzt.
Um diese Behauptung zu belegen, wurde das Testszenario 1 modifiziert. In der dritten Phase wird die Bedienrate unter die minimale Bedienrate von 41,7 kBit/s auf 40 kBit/s gesenkt. Die Diagramme der Bedienraten in der Abbildung A.7 und Warteschlangen-

längen in Abbildung A.8 zeigen nun das erwartete Ergebnis – die Warteschlange wird in Phase drei geleert. Die Paketablehnungsrate in der Abbildung A.10 liegt nun während der vierten Phase deutlich näher an der der Drop Tail Queue. Die Überschreitung ist durch das gewünschte schnellere Wachstum der Warteschlange der Burst Shaping Queue begründet und resultiert nicht mehr aus ihrer partiellen Belegung , wie in Abbildung A.8 ersichtlich ist. Damit zeigt die Burst Shaping Queue das erwartete Verhalten sowohl in der dritten, als auch in der vierten Phase.
Das Verhalten der Burst Shaping Queue in der dritten Phase des Testszenarios fördert die deutliche Schwäche der Implementierung und der zu Grunde liegenden Analysen zu Tage. Sie sind deswegen aber nicht falsch, sondern unvollständig. Das Verhalten der Burst Shaping Queue bei fallenden Warteschlangen wurde nicht genügend berücksichtigt und bedarf daher weiterer Bearbeitung. Eventuelle Lösungsansätze für das Problem werden in Kapitel 6 aufgezeigt. Als direkte Konsequenz werden Situationen, in denen das bekannte Fehlverhalten auftritt, im folgenden Testszenario bewusst vermieden.

5.3. Testszenario 2: Auswirkungen unterschiedlicher Verzögerungsfunktionen

Der folgende Abschnitt stellt das zweite Testszenario und seine Ergebnisse vor und betrachtet diese im Vergleich zu den in Abschnitt 3.4 angestellten Überlegungen.
Im zweiten Testszenario werden die Auswirkungen der im Abschnitt 3.4 erarbeiteten Verzögerungsfunktionen auf Bursts unterschiedlicher Länge untersucht. In der schon bekannten Topologie wird dazu der Verkehrsgenerator für eine bestimmte Zeit aktiviert und deaktiviert. Es entstehen in der Aktivierungszeit Datenströme aus aufeinander folgenden Paketen. Durch die Wahl einer genügend hohen Datenrate folgen diese Pakte kurz hintereinander und überlasten die zu untersuchende Warteschlange. Die Länge, also zeitliche Dauer, des so erzeugten Bursts kann durch die Dauer der Aktivität des Verkehrsgenerators genau bestimmt werden. Damit können identische Umgebungsbedingungen für jeden Simulationslauf ermöglicht werden.
Um unterschiedliche Burstlängen zu untersuchen, wird das Testszenario in drei Teilszenarios, 2a, 2b und 2c, unterteilt. Die Teilszenarios unterscheiden sich nur durch die Länge der erzeugten Bursts sowie die im Teilszenario 2c von 250 kBit auf 300 kBit erhöhte Datenrate. Jedes Teilszenario wird fünfmal für eine Burst Shaping Queue auf Basis einer Drop Tail Queue durchlaufen. In den fünf Läufen werden die Verzögerungsfunktionen aus Abschnitt 3.4 je einmal verwendet.
Die zu beobachtende Warteschlange wird mit den schon aus dem ersten Testszenario bekannten Parametern konfiguriert:

- Warteschlangenlänge: 50 Pakete
- Warteschlangenschwellwert: 35 Pakete
- initiale Verzögerung: 0.002 Sekunden
- maximale Verzögerung: 0.01 Sekunden

Die Größe der zu erzeugenden Paket wird ebenfalls auf 500 Byte festgesetzt. Durch die identische Wahl der initialen Verzögerung liegt die initiale Bedienrate wieder bei 125 kBit/s. Zur besseren Bewertung der Messergebnisse wird jedes Teilszenario ein sechstes Mal mit einer Drop Tail Warteschlange als Referenz durchlaufen. Da außer in tatsächlichen Überlastsituationen der Datenstrom durch eine Drop Tail Warteschlange nicht beeinflusst wird, ist ihr Bedienratengraph deckungsgleich mit dem der Ankunftsgeschwindigkeit. Analog überdecken sich die Graphen von Datenrate und Ankunftsrate.
Das erste Teilszenario betrachtet Bursts der Länge 0.9 s. Sie stellen in diesem Kontext Bursts mittlerer Länge dar und dienen der Überprüfung der Aussagen über die allgemeinen Eigenschaften der Verzögerungsfunktionen. Im nächsten Teilszenario werden kurze Bursts der Länge 0.2 s erzeugt. Das dritte Teilszenario misst dann die Reaktion auf lange Bursts von 1.5 s Dauer. Diese beiden Szenarien prüfen die Annahmen über die besonderen Eignungen der verschiedenen Verzögerungsfunktionen hinsichtlich kurzer und langer Bursts. Die Ruhezeit zwischen den Bursts ist so gewählt, das sich die Warteschlangen komplett leeren können. In allen Subszenarien werden nur zwei oder drei Bursts erzeugt um die Messdaten so übersichtlich wie möglich darstellen zu können.

Dem zweiten Testszenario liegen die Betrachtungen aus Abschnitt 3.4 zu Grunde. Auf dieser Basis sollten sich die Verläufe der Simulationen in den Teilszenarios wie folgt gestalten:
Alle Burst Shaping Queues werden formend auf den Datenverkehr einwirken, d.h sie werden für die Bedienung aller Pakete des Burst länger benötigen als die Drop Tail Warteschlange. Ihre Bedienrate wird später ansteigen und ebenfalls später abfallen. Gleichzeitig werden die Warteschlangen aller Verzögerungsfunktionen schneller wachsen als die der Drop Tail Queue. Ihre Bedienraten werden kleiner als die der Drop Tail Queue sein, wenn ihre Warteschlange weniger als 35 Pakete enthält. Beim Erreichen und Überschreiten dieses Wertes werden sie genauso schnell bedienen wie die Drop Tail Queue. Die Bedienrate von $f_0(l)$ wird im Vergleich zu den anderen Verzögerungsfunktionen am gleichmäßigsten wachsen und fallen. Eine Warteschlange mit dieser Verzögerungsfunktion wird ein sehr ausgeglichenes Verhalten bei jeglicher Burstlänge zeigen. Sie wird bei allen Burstlängen ein relativ gutes Verhalten zeigen, in den Spezialfällen kurzer oder langer Bursts aber durch andere Bedienfunktionen übertroffen werden. Als Ausgleich dazu wird sie aber auch keine ausgeprägten Schwächen im Verhalten gegenüber dem Datenverkehr zeigen.
Die Bedienraten von $f_2(l)$ und $f_4(l)$ werden bei kleinen Warteschlangenlängen schneller wachsen als die von $f_0(l)$, bei Warteschlangenlängen nahe dem Schwellwert dagegen langsamer. Ihre Warteschlangen werden deshalb weniger stark anwachsen als die von f_0. Dieses Verhalten wird bei $f_4(l)$ ausgeprägter sein als bei $f_2(l)$. Ihr Einfluss auf den Datenverkehr wird am ehesten dem einer Drop Tail Queue ähneln. Die Folge ist ein besonders gutes Formungsverhalten bei längeren Bursts, da Warteschlangen mit diesen Funktionen auf Grund des geringeren Längenwachstums sehr spät überlaufen. Kurze Bursts werden dagegen relativ wenig beeinflusst – Warteschlangen mit diesen Verzögerungsfunktionen verhalten sich hier im Vergleich zu den anderen am schlechtesten.
Im Gegensatz dazu werden die Bedienraten von $f_1(l)$ und $f_3(l)$ bei niedrigen Warteschlangenlängen langsamer wachsen als die von f_0, bei Längen nahe dem Schwellwert dagegen schneller. Daher werden ihre Warteschlangen auch insgesammt am stärksten anwachsen. Diese Eigenschaften sind dabei bei $f_3(l)$ ausgeprägter als bei $f_1(l)$. Sie zeigen insbesondere bei kurzen Bursts günstiges Verhalten, da das Datenaufkommen am meisten geglättet wird. Lange Bursts hingegen lassen Warteschlangen mit diesen Verzögerungsfunktionen im Vergleich zu den anderen schneller überlaufen – sie zeigen hier das schlechteste Verhalten aller getesteten Verzögerungsfunktionen.
Die Ergebnisse der Simulationen werden in den Diagrammen in den Anhängen B bis D dargestellt. Für folgende Messgrößen wird der Verlauf über die Simulationszeit dargestellt:

- Abbildung B.1, C.1, D.1 Bedienrate
- Abbildung B.2, C.2, D.2 Warteschlangenlänge
- Abbildung B.4, C.4, D.5 Differenz aus Ankunfts- und Bedienrate
- Abbildung B.3, C.3, D.3 Datenrate
- Abbildung D.4 Abgelehnte Pakete

Die abgelehnten Pakete für die Subszenarien 2a und 2b werden nicht dargestellt, da hier durch die Wahl identischer Ankunfts- und Linkrate keine Paketverluste auftreten konnten. Die Graphen der Messdaten bestätigen das erwartete Verhalten der Burst Shaping Queue in diesem Testszenario deutlich.
Im ersten Teilszenario werden Bursts mittlerer Länge generiert. Die Drop Tail Warteschlange folgt wie erwartet dem Eingangsdatenstrom. Dies ist am Verlauf des Graphen der Differenz von Ankunfts- und Bedienrate in Abbildung B.4 erkennbar, der über die gesamte Simulationszeit praktisch nicht vom Wert 0 abweicht. Da die Drop Tail Warteschlange mit einer Linkrate von 250 kBit/s bedient, wird sie nicht überlastet. Die verschiedenen Verzögerungsfunktionen reagieren ebenfalls wie erwartet auf Überlast. Das Anwachsen der Warteschlange und das eigentlich unerwartete Abweichen der Differenz von Ankunfts- und Bedienrate von 0 ist mit einer Eigenart des CBR-Verkehrsgenerators zu erklären. Die ersten beiden Pakete werden mit Linkrate, d.h. 500 kBit/s erzeugt und sorgen damit für tatsächliche Überlast am Anfang des ersten Bursts. Beim zweiten Burst trat dieses Verhalten nicht mehr auf.
Die Bedienraten von $f_2(l)$ und $f_4(l)$ wachsen erwartungsgemäß bei niedrigen Warteschlangenlängen sehr schnell und mit wachsender Warteschlangenlänge immer langsamer. Nach Ende des Bursts fallen die Bedienraten bei diesen Verzögerungsfunktionen schnell ab. Ihre Warteschlangenlängen sind während des Bursts am kürzesten und wachsen bei zunehmender Warteschlangenlänge langsamer als die der anderen Verzögerungsfunktionen. Nach anfänglich steilem Ansteigen der Bedienrate zeigt der Graph von $f_2(l)$ ein kurzes Abfallen der Bedienrate, die danach wie geschildert weiter wächst. An diesem Graphenverlauf ist deutlich die Wirkung des Burst Shaping Algorithmus zu erkennen. An diesem Zeitpunkt beginnt die Warteschlange zu wachsen und die Beeinflussung der Bediengeschwindigkeit setzt ein. Nach Ende des Bursts wird die Warteschlange bei diesen Funktionen am schnellsten geleert. Die Datenraten beider sind genau wie erwartet am höchsten, d.h. ihre Graphen haben den geringsten Abstand zu dem der Drop Tail Warteschlange. Ihr Formungsverhalten ist ebenfalls erwartungsgemäß das geringste aller Verzögerungsfunktionen. Die Graphen der Differenz von Ankunfts- und Bedienrate zeigen ein sehr steiles Ansteigen und Abfallen. Sie weichen die geringste Zeit vom Graph der Drop Tail Warteschlange ab. Der Burst wird also am wenigsten verformt. Dabei zeigt $f_4(l)$ das geschilderte Verhalten erwartungsgemäß etwas ausgeprägter als $f_2(l)$.
Im Gegensatz dazu steigen die Bedienraten von $f_1(l)$ und $f_3(l)$ erst relativ spät an und steigen dann sehr schnell auf die Linkrate. Nach Ende des Bursts fallen sie anfangs steil ab. Mit weiter abnehmender Warteschlangenlänge wird die Bedienrate in Abbildung B.1 wie erwartet immer langsamer und wird bis zur Leerung der Warteschlange weiter vermindert. Der steile Abfall am Ende des Abbaus der Warteschlange ist mit dem in Abschnitt 3.2 geschilderten Verhalten zu erklären – das letzte Paket wird im Vergleich zum vorletzten sehr schnell bedient, d.h. mit initialer statt maximaler Bedienrate. Entsprechend dem schnellen Wachstum der Bedienrate erreichen die Warteschlangen mit diesen Verzögerungsfunktionen sehr schnell ihre maximale, in diesem Fall durch den Schwellwert begrenzte, Länge. Weiteres Wachstum ist auf Grund der Ankunftsrate von 250 kBit/s nicht möglich – die Warteschlange befindet sich genau in Grenzlast. Nach Ende des Bursts werden die Warteschlangen ebenfalls wie erwartet relativ langsam geleert. Die Datenraten der Warteschlangen mit den Verzögerungsfunktionen $f_2(l)$

und $f_4(l)$ liegen genau wie erwartet unter allen anderen. Beide Verzögerungsfunktionen zeigen im Graphen der Differenz von Ankunfts- und Bedienrate in Abbildung B.4 das stärkste Formungsverhalten, erkennbar an ihrer Abweichung vom Graphen der Drop Tail Queue. Dieses Verhalten entspricht den Annahmen über ihr Verhalten. Ebenfalls erwartungskonform ist die stärkere Ausprägung der geschilderten Eigenschaften bei Warteschlangen mit Verzögerungsfunktion $f_3(l)$ als bei Warteschlangen mit Verzögerungsfunktion $f_1(l)$.
Die Warteschlange mit Verzögerungsfunktion $f_0(l)$ zeigt gleichermaßen ein ausgeglichenes Verhalten. Wachstum und Abfallen ihrer Bedienrate liegen wie angenommen zwischen den Graphen der anderen Verzögerungsfunktionen. Ihre Warteschlangenlänge wächst und fällt ebenfalls wie angenommen relativ ausgeglichen. Die Datenrate der Warteschlange mit dieser Verzögerungsfunktion bewegt sich nahe dem arithmetrischen Mittel der anderen Funktionen. Ihr am Graphen der Differenz von Ankunfts- und Bedienrate in Abbildung C.4 erkennbares Formungsverhalten ist in Relation zu den anderen Funktionen ebenfalls ausgeglichen.
Im Teilszenario 2b zeigen die Verzögerungsfunktionen sehr deutlich das vermutete Verhalten bei kurzen Bursts. Besonders verdeutlicht wird hier die gute Eignung von Warteschlangen mit den Verzögerungsfunktionen $f_3(l)$ und $f_1(l)$ in diesem Spezialfall. Sie zeigen ein sehr gutes Verformungsverhalten, erkennbar am sehr gleichmäßigen und langsamen Abnehmen der Bedienrate nach Ende des Bursts bei gleichzeitig hoher Warteschlangenlänge. Der durch den Burst verdichtete Datenstrom wird relativ gleichmässig auf einen größeren Zeitraum als die ursprüngliche Burstzeit verteilt. Der entstehende Paketstrom ist durch die relativ langsam fallende Bedienrate bei Abbau der Warteschlange deshalb sehr glatt. Die Graphen von $f_2(l)$ und $f_4(l)$ glätten bei weitem nicht so gut, da ihre Bedienrate schneller fällt und ihre Warteschlangen nicht so stark anwachsen. Daher erzeugen sie einen Datenstrom, der nur wenig glatter ist und gleichzeitig geringere Paketabstände besitzt. Diese Verzögerungsfunktionen zeigen hier das erwartet weniger günstige Verhalten. Die Graphen der Warteschlange mit Verzögerungsfunktion $f_0(l)$ liegen wie erwartet im Mittel der anderen.
Im Teilszenario 2c zeigen alle Verzögerungsfunktionen ebenfalls das angenommene Verhalten. Hier werden die guten Eigenschaften von $f_2(l)$ und $f_4(l)$ beim Verarbeiten langer Bursts sehr deutlich – sie sind die einzigen Verzögerungsfunktionen, bei denen die Warteschlange keine Pakete verwerfen muss und trotzdem den ankommenden Burst glättet. Dieses gute Verhalten bei tatsächlicher Überlast wird nur durch die Drop Tail Warteschlange übertroffen, die aber den ankomenden Burst nicht formt. Die anderen Verzögerungsfunktionen zeigen in dieser Situation ihre Leistungsgrenzen. Sie glätten den Datenstrom zwar stärker als $f_1(l)$ und $f_3(l)$ müssen aber auf Grund überlaufender Warteschlangen Pakete verwerfen. Damit verhalten sie sich deutlich schlechter als die beiden anderen Verzögerungsfunktionen.

6. Zusammenfassung, Bewertung und Ausblick

In diesem letzten Kapitel der Arbeit wird einführend ein kurzer Überblick über die Ergebnisse der vorangegangenen Kapitel gegeben. Diese werden im folgenden zusammengefasst und die Ergebnisse der Arbeit bewertet. Aus dieser Bewertung wird eine Gesamtbewertung des Ansatzes gewonnen sowie ein Überblick über notwendige Verbesserungen und weitere, noch offene Fragen gegeben.

6.1. Zusammenfassung und Bewertung

Innerhalb dieser Arbeit wurde die Burst Shaping Queue von einer relativ groben Funktionsbeschreibung hin zu einer konkreten, nutzbaren Implementierung entwickelt und erste Simulationen zur Überprüfung der Analyseergebnisse durchgeführt. Dieser Prozess wird hier in seinen wichtigsten Schritten und Ergebnissen noch einmal zusammengefasst und bewertet.
Im Kapitel 2 schildert das allgemeine Umfeld paketvermittelnder Netze und erläutert das Problem von Überlast in Folge von kurzzeitiger Überlast durch Bursts. Die Burst Shaping Queue wurde als ein Ansatz zur Vermeidung solcher Situationen vorgestellt. Überlast in Folge von Bursts soll in diesem Ansatz durch eine bessere Nutzung der Warteschlangenkapazitäten auf dem Übertragungsweg und eine Glättung des Datenverkehrs vermieden werden.
Das folgenden Kapitel 3 betrachtete die Problemstellung grundsätzlich. Dazu erfolgte als erstes in Abschnitt 3.1 eine Definition des Burst-Begriffes. Danach wurde die Lastsituation an Warteschlangen analysiert und durch Einführung der Begriffe Normallast, Grenzlast und Überlast präzisiert. Es folgte die Erörterung der Möglichkeiten der Beeinflussung der Bedienrate einer Warteschlange. Auf Grund der gegebenen festen Linkrate schied die Möglichkeit einer direkten Manipulation der Bedienrate aus. Als Alternative dazu entstand das Konzept der Verzögerung zur Beeinflussung der effektiven Bedienzeit für ein Paket. Dieses Konzept führte zu einem ersten Entwurf eines Algorithmus, der die Bedienrate der Warteschlange durch Berechnung von Verzögerungen für Pakete in Abhängigkeit der aktuellen Warteschlangenlänge reguliert. Der Erläuterung dieses Grobentwurfs schloss sich eine Analyse von Warteschlangen an, die zeigte, dass es die Burst Shaping Queue als solche nicht gibt. Das Konzept stellt vielmehr eine erweiterte, von der Paketauswahlstrategie unabhängige Eigenschaft von Warteschlangen dar. Folglich ist sie beliebig mit diesen kombinierbar. Die Menge der zu betrachtenden Strategien wurde danach auf einen für diese Arbeit angemessenen Rahmen beschränkt und

das Drop Tail Verfahren als Basis der Implementierung ausgewählt.
Der nächste Abschnitt 3.2 begann mit der Einordnung des Burst Shaping Ansatzes in die Klasse der Closed Loop Ansätze zur Überlastvermeidung in paketvermittelnden Netzwerken. Darauf basierend wurde eine erste Abschätzung der voraussichtlichen Leistungsfähigkeit des Ansatzes gegeben und die Grenzen seiner Möglichkeiten aufgezeigt. Aus einer genaueren Betrachtung der Puffernutzung des Burst Shaping Ansatzes entstand die Notwendigkeit, den eingeführten Begriff der Überlast zu verfeinern. Die initiale Bediengeschwindigkeit wurde eingeführt und Überlast in Folge der Überschreitung dieser durch die Ankunftsrate von tatsächlicher Überlast durch Überschreitung der Linkrate abgegrenzt. Damit konnte das genaue Verhalten des Burst Shaping Ansatzes in verschiedenen Lastsituationen beschrieben werden. Als Ausgangspunkt zur weiteren Entwicklung der Burst Shaping Ansatzes wurde die weit verbreitete Drop Tail Queue festgelegt. Ihr Verhalten im Netzwerk bildet gleichzeitig den Maßstab für den Erfolg des Burst Shaping Ansatzes. Der Burst Shaping Ansatz kann nur dann erfolgreich sein, wenn er das Verhalten der Drop Tail Warteschlange bei kurzer Überlast im Kontext des gesamten Netzwerkes verbessert. Gleichzeitig darf sich sein zwingend schlechteres Verhalten bei unvermeidlichen Paketverlusten nicht entscheidend auf die Gesamtheit des Netzwerkes auswirken. Den Abschluss dieses Abschnittes bildete eine Erörterung der Auswirkungen der Verformung des Verkehrsprofils auf die Verbindungsparameter Delay und Jitter.
In den folgenden beiden Abschnitten 3.3 und 3.4 wurden die formalen Rahmen für Funktionen, die eine Abhängigkeit von Verzögerung und Warteschlangenlänge abbilden, definiert. Danach erfolgte eine erste Einteilung in Funktionsgruppen mit voraussichtlich unterschiedlichen Auswirkungen auf den Ausgangsdatenstrom und die Auswahl repräsentativer Kandidaten für die konkrete Implementierung. Ihr voraussichtlicher Einfluss auf den Ausgangsdatenstrom und daraus resultierende Einsatzszenarien schlossen dieses Kapitel ab.
Kapitel 4 begann in Abschnitt 4.1 mit einer kurzen Vorstellung des Zielsystems der Implementierung, dem Netzwerksimulator ns2. Im folgenden Abschnitt 4.2 wurde die Modellierung von Wartschlangenmechanismen im ns2 ausführlich analysiert. Auf Basis dieser Analyse erfolgt in Abschnitt 4.3 eine Diskussion der Modellierungsmöglichkeiten der Implementierung der Burst Shaping Eigenschaft. Unter Nutzung der unter dem Gesichtspunkt objektorientierten Softwareentwurfs guten Modellierung des ns2 war es möglich, die Burst Shaping Eigenschaft durch Einführung einer einzigen abstrakten Basisklasse umzusetzen. Der folgende Abschnitt 4.4 beschrieb dann die innere Struktur der Klasse, den konkreten Algorithmus der Burst Shaping Eigenschaft und seine Implementierung. Darüber hinaus wurden weitere, neben der eigentlichen Klassenimplementierung notwendige Schritte bis zur Nutzbarkeit der Implementierung in ns2- Simulationen beschrieben. Der letzten Abschnitt 4.6 erläuterte Entscheidungen zur Einschränkungen der Implementierung im Kontext der Aufgabenstellung und bewertet diese. Dies betraf die Einschränkung des verwendeten Maßes der Warteschlangenlänge (Paketanzahl) sowie Einschränkungen in der intuitiven Verwendbarkeit der Implementierung durch Dritte durch fehlende Accessormethoden. Das Ende des Abschnittes zeigte dann Möglickeiten der Verbesserung und Umsetzungsvorschläge auf.
Das Kapitel 5 schließlich begann mit der Erläuterung der Testziele und der. Die Testtopologie wurde vorgestellt und ein kurzer Überblick über die konkreten Simulations-

skripte gegeben. Der folgende Abschnitt 5.2 stellte dann das erste Szenario zum Testen der Implementierung vor. Es wurden auf Basis vorangegangener Analysen Voraussagen über das Verhalten des Burst Shaping Ansatzes getroffen und an Hand der Messwerte überprüft. Dabei stellte sich eine konzeptionelle Lücke in der Problemanalyse heraus. Das Verhalten der Warteschlange bei fallenden Warteschlangenlängen war nicht genug betrachtet worden. Die Folge waren unnötige Warteschlangenbelegungen die unnötige Paketverluste bewirkten. Um den Einfluss des Fehlers auf die Testresultate auszuschliessen erfolgte eine Modifikation des ersten Testszenarios und eine erneute Messung, die dann das erwartete Verhalten des Burst Shaping Ansatzes bestätigte. Die Erörterung von Lösungsmöglichkeiten wurde auf Kapitel 6 verschoben. Der folgende Abschnitt stellte dann das zweite Testszenario zum Verhalten der verschiedenen Verzögerungsfunktionen aus 3.4 bei verschieden langen Bursts vor. Die Messdaten dieses Szenarios bestätigten das erwartete Verhalten voll.
Als Ergebnis der Arbeit lässt sich nun folgendes festhalten: Die Burst Shaping Queue stellt nach dem ersten gewonnen Eindruck einen verfolgenswerten Ansatz zur Lösung des Problems von kurzzeitiger Überlast in paketorientierten Netzwerken dar. Die angestellten Analysen zum allgemeinen Verhalten konnten in ersten Simulationen belegt werden, stellten sich aber in einem Spezialfall als unvollständig heraus. Die existierenden Einschränkungen der Implementierung des Ansatzes können nach Ansicht des Autors leicht behoben werden. Damit steht nach Beseitigung der konzepionellen Unvollständigkeit und einer entsprechenden Änderung der Implementierung als Resultat dieser Arbeit eine solide Basis für eine weitere Bearbeitung des Themas durch Dritte zur Verfügung. Bei weiteren positiven Ergebnissen, die auf einen hohen Nutzwert des Konzeptes schließen lassen, könnte eine offizielle Integration des Burst Shaping Ansatzes in den ns2 eine breite Basis für eine breite Bearbeitung des Themas schaffen.

6.2. Ausblick

Ziel dieser Arbeit sind grundsätzliche Aussagen zum Burst Shaping Ansatz, eine erste Implementierung und Überprüfung der Analyseergebnisse in einfachen Tests. Die Bestätigung der grundsätzlichen positiven Eigenschaften der Burst Shaping Queue und die Aufdeckung von Lücken im analytischen Fundament bietet nun die Möglichkeit weiterer Arbeiten zu diesem Thema.
Erste und wichtigste Aufgabe sollte die Behebung des unzulänglichen Verhaltens bei fallenden Warteschlangenlängen sein. Durch die Beobachtung der Warteschlangenentwicklung oder der Ankunftsrate kann ein Fallen der Warteschlange erkannt werden. Mit dieser Information ist es möglich, die Bediengeschwindigkeit des Burst Shaping Ansatzes bei ausreichend geringer Ankunftsrate so zu regulieren, dass die Warteschlange geleert wird. Dabei ist darauf zu achten, das die Leerung keine Bursts im Ausgangsdatenstrom erzeugt. Dies könnte folgendermaßen erreicht werden:
Wenn die Bediengeschwindigkeit der Warteschlange die Ankunftsrate erreicht und diese kleiner als die initiale Bediengeschwindigkeit ist, könnte die Neuberechnung der aktuellen Bediengeschwindigkeit gestoppt und die Warteschlange mit initialer Geschwindigkeit bis zur Leerung bedient werden werden. Es entsteht so ein gleichmäßiger Datenstrom ohne Peaks. Die Funktionalität der Burst Shaping Queue bliebe voll erhalten

und eine Einschränkung der Qualität des Ausgangsdatenstroms ist nicht zu erwarten, da die Warteschlange nur Datenverkehr mit einer Ankunftsrate über der initialen Bediengeschwindigkeit formt. Allerdings ist der Erfolg dieses Vorgehens auf initiale Bedienraten größer als Null beschränkt. Alternativ könnte statt der initialen Geschwindigkeit auch eine Geschwindigkeit gewählt werden, die leicht oberhalb der aktuellen Ankunftsrate liegt. Der Effekt wäre hinsichtlich der Glättung des Datenverkehrs zwar besser, gleichzeitig ist dies aber mit einer längeren Dauer für die Leerung der Warteschlange verbunden.
Darüber hinaus ist es sicherlich lohnenswert, die schon vorgeschlagenen Änderungen zur Verbesserung der Benutzbarkeit der Implementierung vorzunehmen. Konkret bedeutet das die Schaffung von oTcl- Accessormethoden anstatt dem einfachen Binden von Variablen für die Parameter der Warteschlange. Der Nutzen wäre eine effiziente und sichere Überprüfung von Fehleingaben, die momentan nicht erfolgt. Die Messung der Warteschlangenlänge in Bytes zur Bestimmung der Verzögerung ist eine weitere, einfach umzusetzende Verbesserung. Sie ermöglicht eine Nutzung der Queue in Netzen mit variabler Paketgröße, in denen die momentane Implementierung auf Grund der Annahme konstanter Paketgrößen nicht korrekt funktioniert. Die Vaterklasse der Implementierung, *Queue*, implementiert schon die Längenbestimmung der Warteschlange in Bytes inklusive einem Steuerflag. Die Übertragung dieser Funktion auf die Klasse *QueueBS* ist damit relativ leicht. Desweiteren könnten die initiale und minimale Bediengeschwindigkeit als neue Parameter eingeführt werden und so eine Alternative zum direkten Setzen der initialen und maximalen Verzögerung bieten. Die Werte der initialen und maximalen Verzögerung sind bei vorgegebenen initialen und minimalen Bedienraten aus den Linkraten errechenbar. Der eigentliche, mit Verzögerungen operierende Algorithmus bliebe damit unverändert. Diese Erweiterungen machen die vorliegende Implementierung dann einfacher und komfortabler durch Dritte nutzbar und erleichtern so eine weitere Erforschung des Konzeptes.
Abgesehen von der Implementierung sind noch viele andere Fragen offen. Um genauere Aussagen zum Nutzen des Konzeptes zu machen, müssen viele andere Einflussfaktoren, die in den hier angestellten Simulationen vernachlässigt wurden, in Betracht gezogen werden. So wurde durch die hier angestellten Tests nur die Glättung des Datenverkehrs betrachtet. Simulationen, die die angestrebte bessere Puffernutzung auf dem Übertragungsweg belegen, sind noch nicht erfolgt. Es wurden ebenfalls keine Simulationen mit agregierten Datenströmen aus mehreren Datenquellen, TCP Datenströmen oder selbstähnlichem Datenverkehr, wie er in den Backbonenetzen des Internets anzutreffen ist, durchgeführt. Der Burst Shaping Ansatz wurde in dieser Arbeit auch nur in Kombination mit Drop Tail Warteschlangen betrachtet. Die Simulation des Burst Shaping Ansatzes in Kombination mit RED Strategien erscheint hierbei besonders sinnvoll, da sich beide Ansätze zur Überlaststeuerung durch eine nun gleichzeitig mögliche Beeinflussung von Bedien- und Ankunftsrate effektiv ergänzen könnten.
Neben der allgemeinen Nutzenaussage ist die weiterhin die Bestimmung geeigneter Werte für Warteschlangenlängen und Schwellwerte in Abhängigkeit der Linkraten der Queue notwendig. Die Festlegung dieser Werte erfolgte hier willkürlich und durch simple Tests. Für beide existieren keine gesicherten Erfahrungswerte. Ebenso ist die Bewertung der Eigenschaften geeigneter Verzögerungsfunktionen noch nicht abgeschlossen und bedarf weiterer Betrachtung. Neben Erfahrungswerten könnten Schwellwert,

initiales Delay und Verzögerungsfunktion auch adaptiv in Abhängig vom Eingangsdatenstrom bestimmt werden, um den Einfluss des Burst Shaping Ansatzes weiter zu optimieren. Solche intelligenten Warteschlangen könnten auf lange Sicht zu einer entscheidenden Verbesserung der Verbindungsqualität in den heutigen, paketvermittelnden Netzwerken beitragen.

Literaturverzeichnis

[1] The network simulator ns2. http://www.isi.edu/nsnam/dist/ns-allinone-2.26.tar.gz.

[2] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. 1(4):397–413, 1993.

[3] K. VARADHAN (Editors) K. FALL. The ns manual (online). http://www.isi.edu/nsnam/ns/doc.

[4] D. BLACK K. RAMAKRISHNAN, S. FLOYD. Rfc 3168 the addition of explicit congestion notification (ecn) to ip. http://rfc-editor.org/rfc/rfc3168.txt, Sep. 2001. Status: Experimental.

[5] W. STEVENS M. ALLMAN, V. PAXSON. Rfc 2581 tcp congestion control. http://rfc-editor.org/rfc/rfc3168.txt, Apr. 1999. Status: Proposed Standard.

[6] A. S. TANENBAUM. Computernetzwerke, 1998.

Abbildungsverzeichnis

A. Messdatendiagramme der Testszenario 1

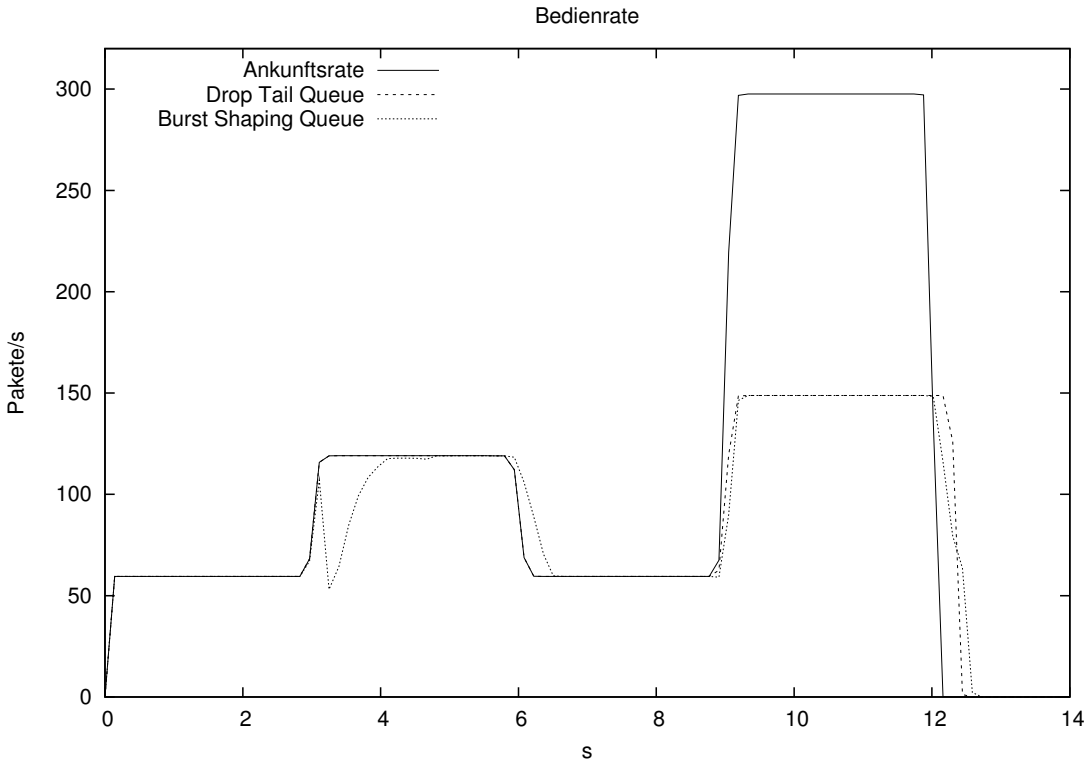

Abbildung A.1.: Bedienraten Testszenario 1

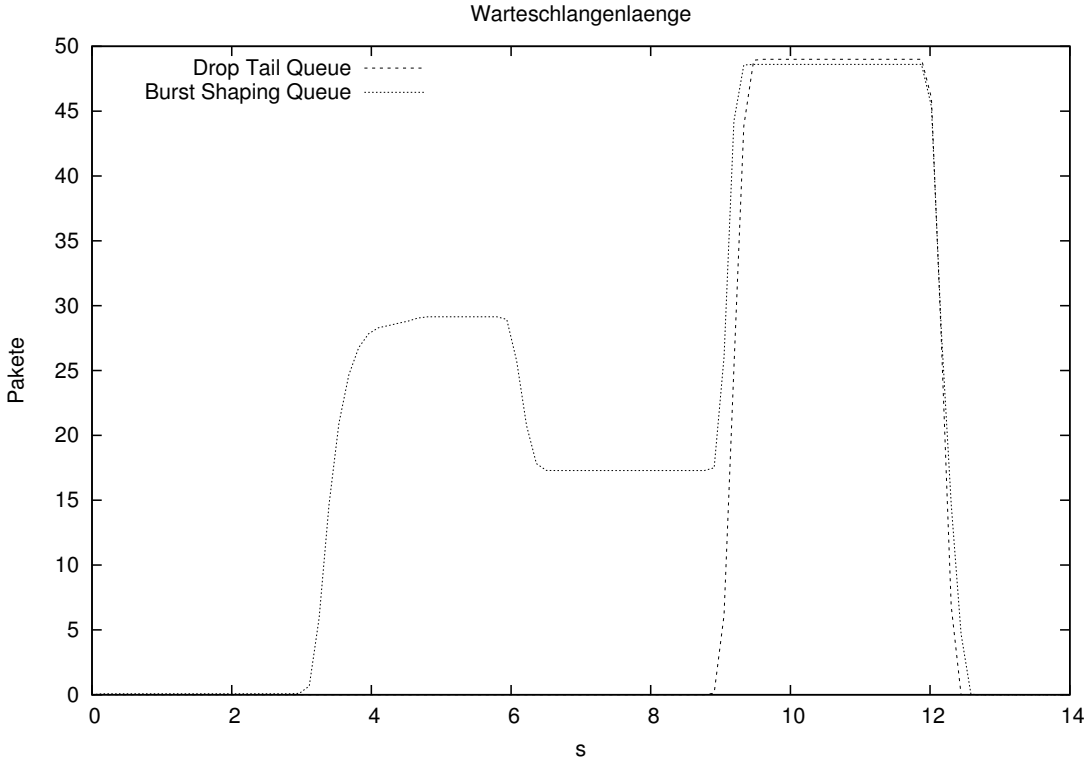

Abbildung A.2.: Warteschlangenlänge Testszenario 1

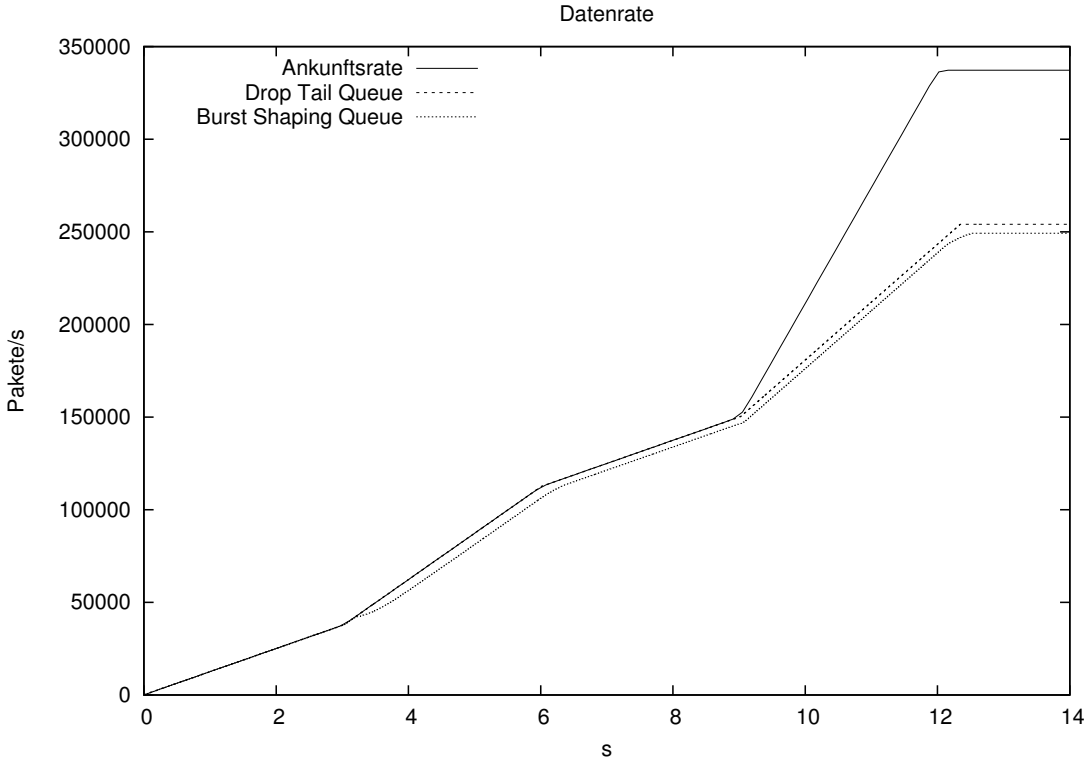

Abbildung A.3.: Datenrate Testszenario 1

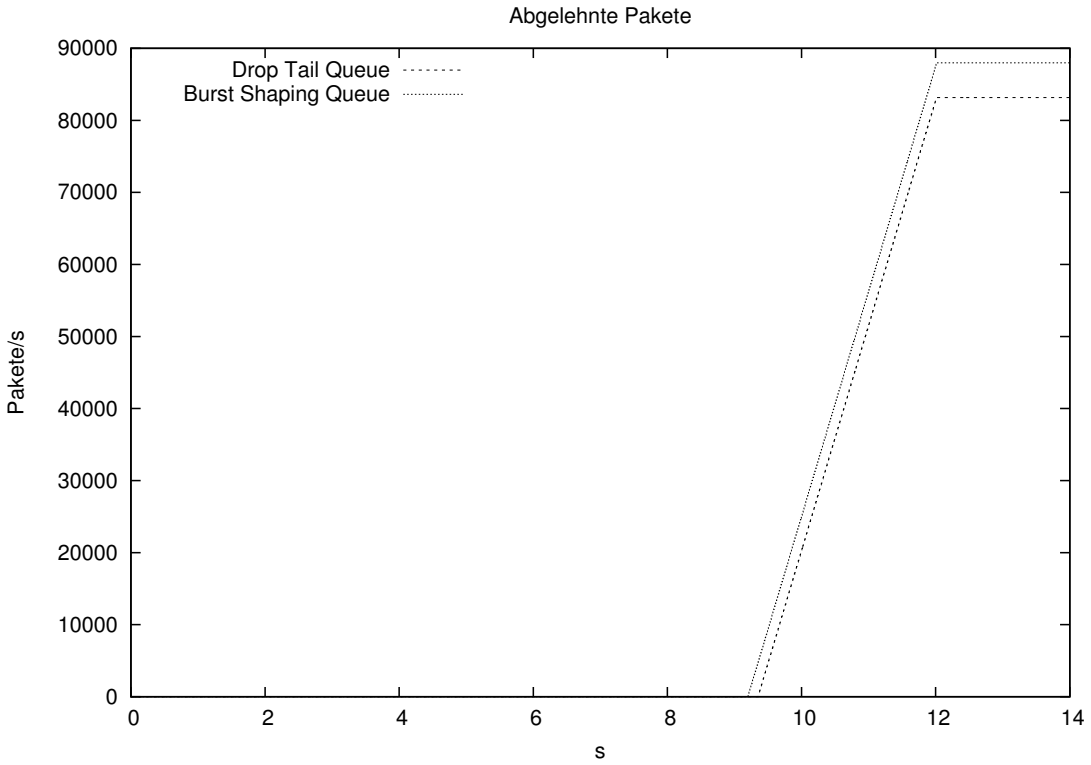

Abbildung A.4.: Abgelehnte Pakete Testszenario 1

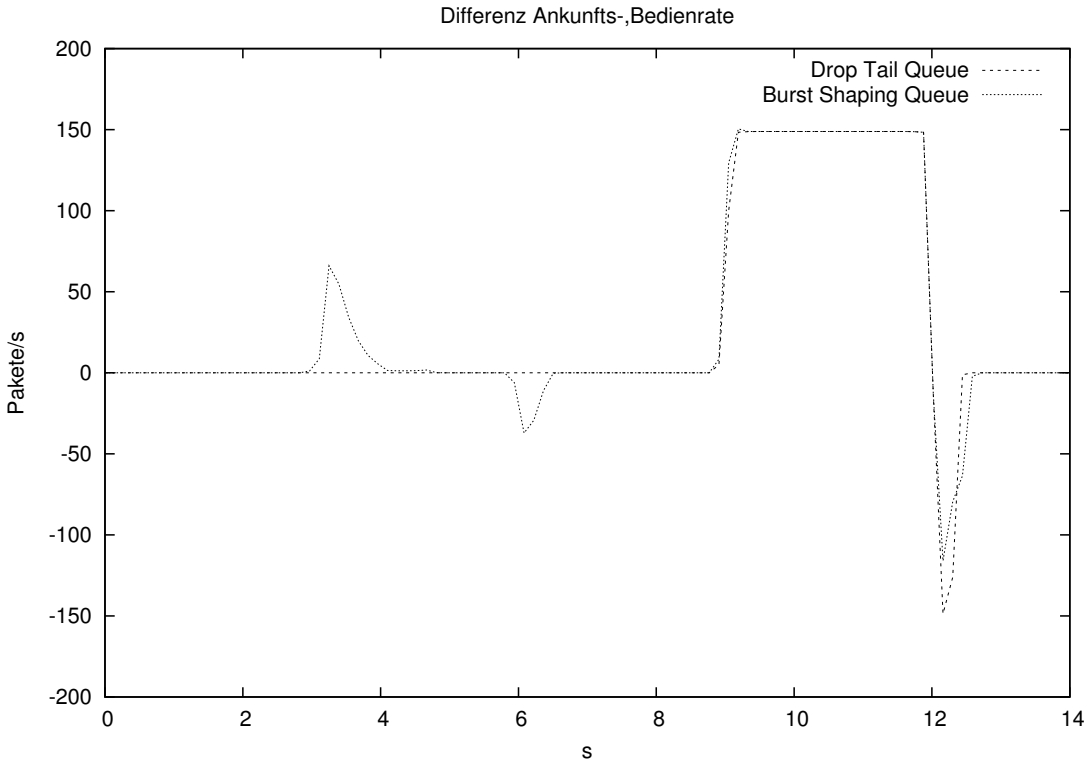

Abbildung A.5.: Differenz Ankunfts-/ Bedienrate Testszenario 1

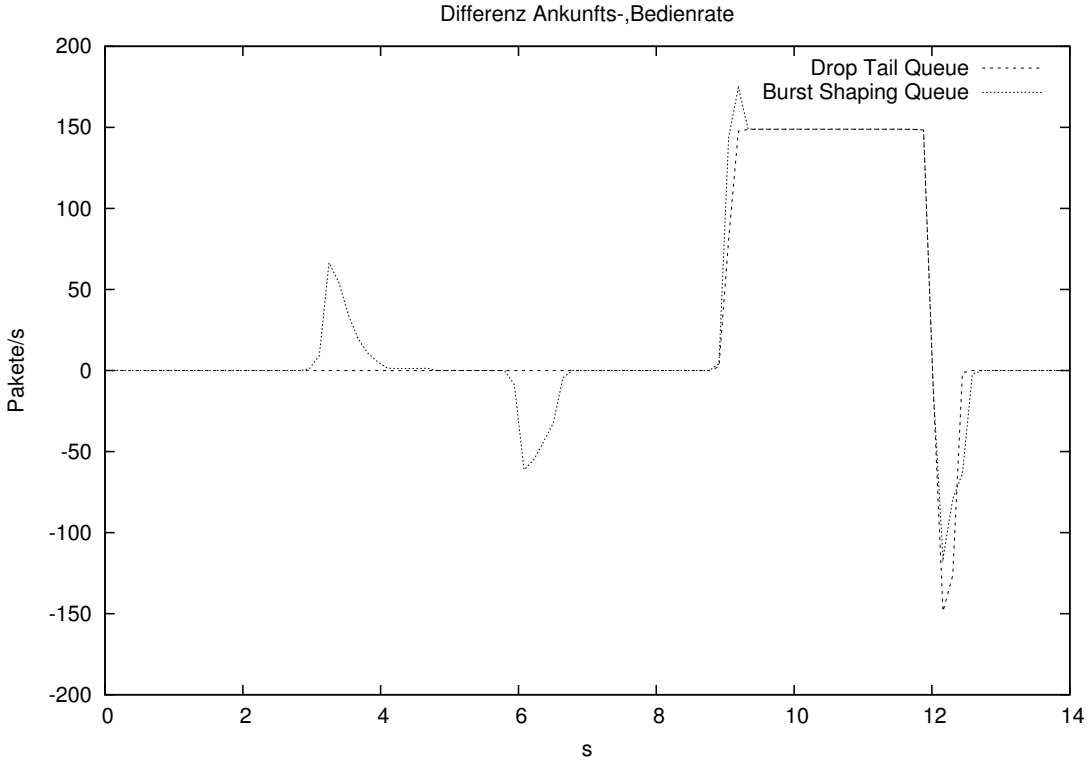

Abbildung A.6.: Differenz Ankunfts-/ Bedienrate Testszenario 1 (modifiziert)

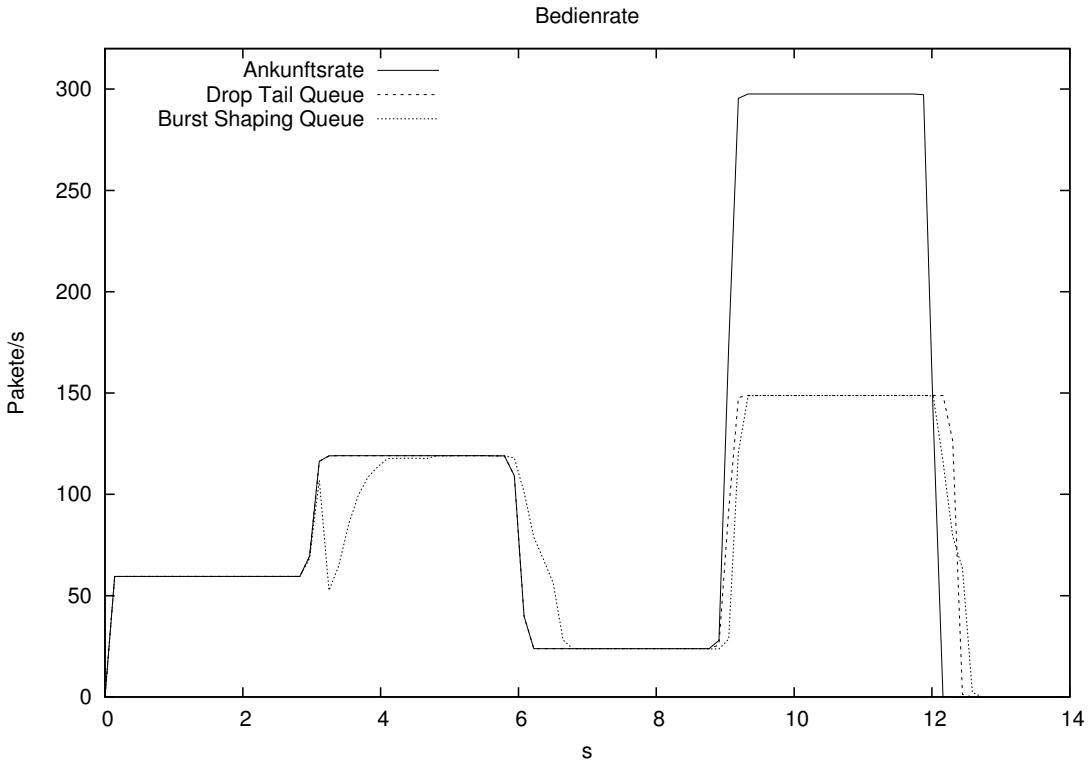

Abbildung A.7.: Bedienraten Testszenario 1 (modifiziert)

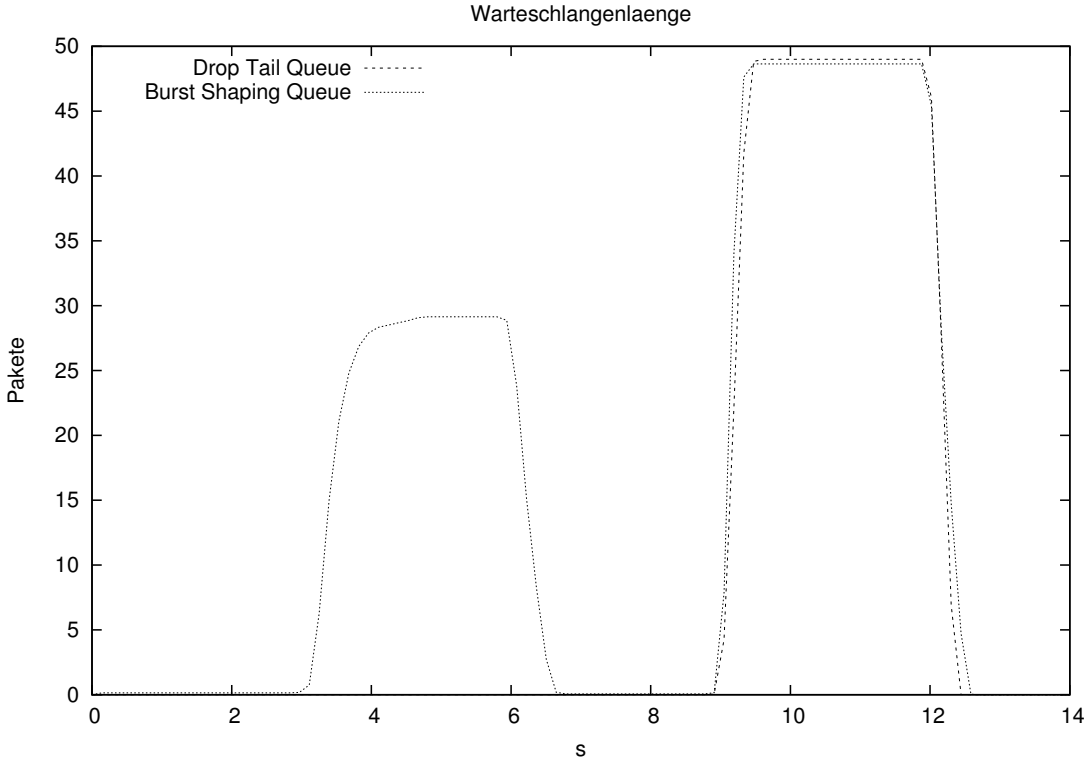

Abbildung A.8.: Warteschlangenlänge Testszenario 1(modifiziert

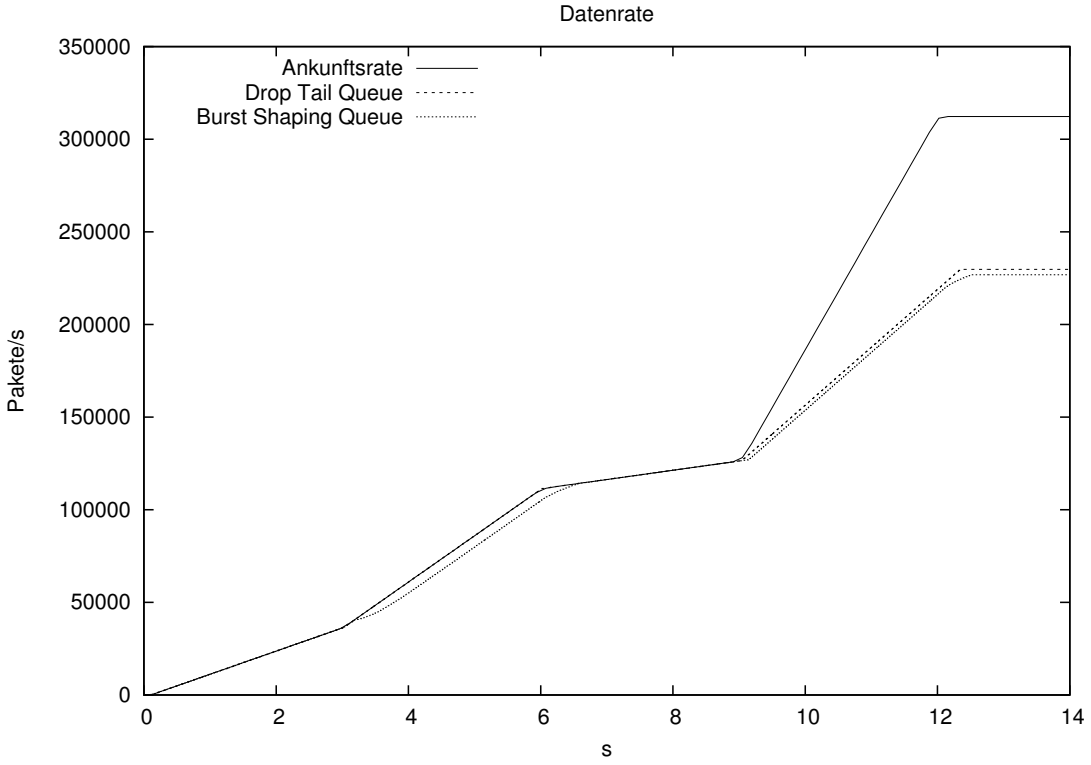

Abbildung A.9.: Datenrate Testszenario 1 (modifiziert)

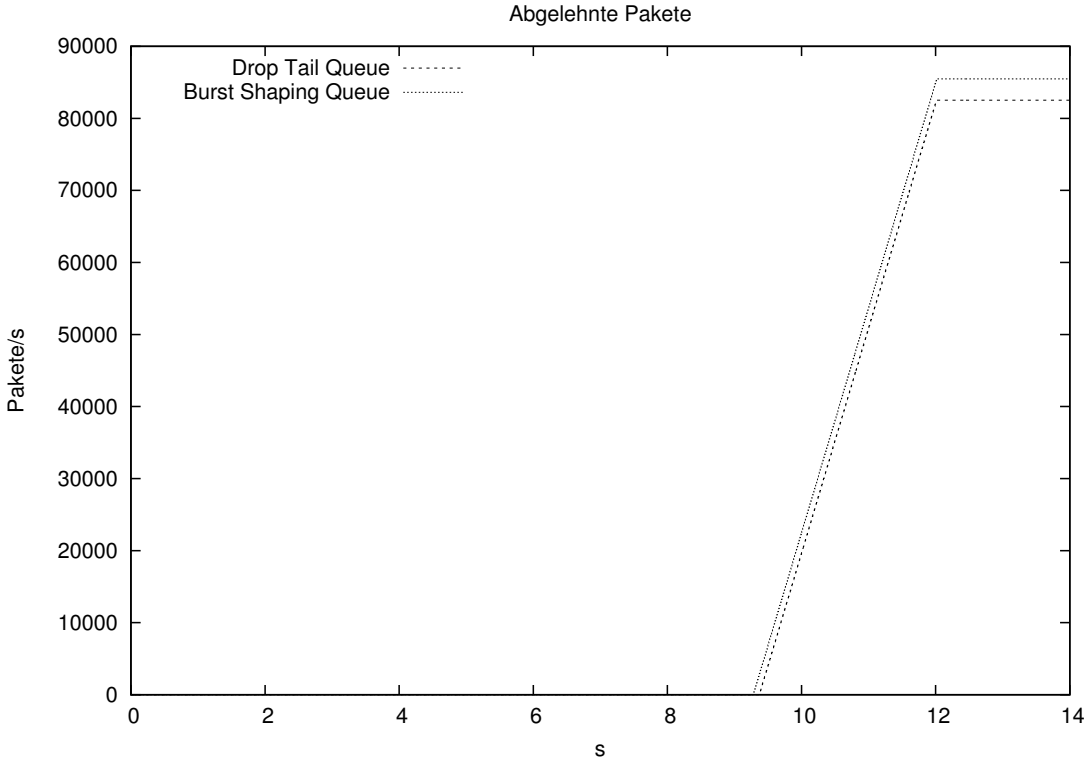

Abbildung A.10.: Abgelehnte Pakete Testszenario 1 (modifiziert)

B. Messdatendiagramme Testszenario 2a

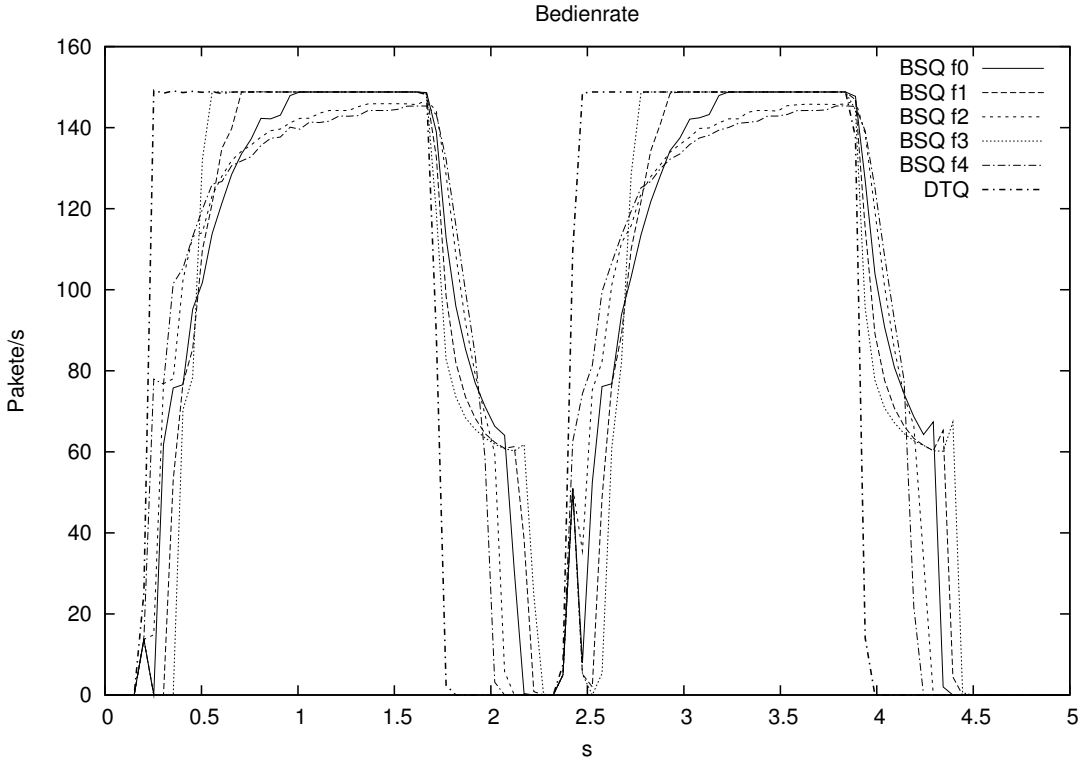

Abbildung B.1.: Bedienraten Testszenario 2a

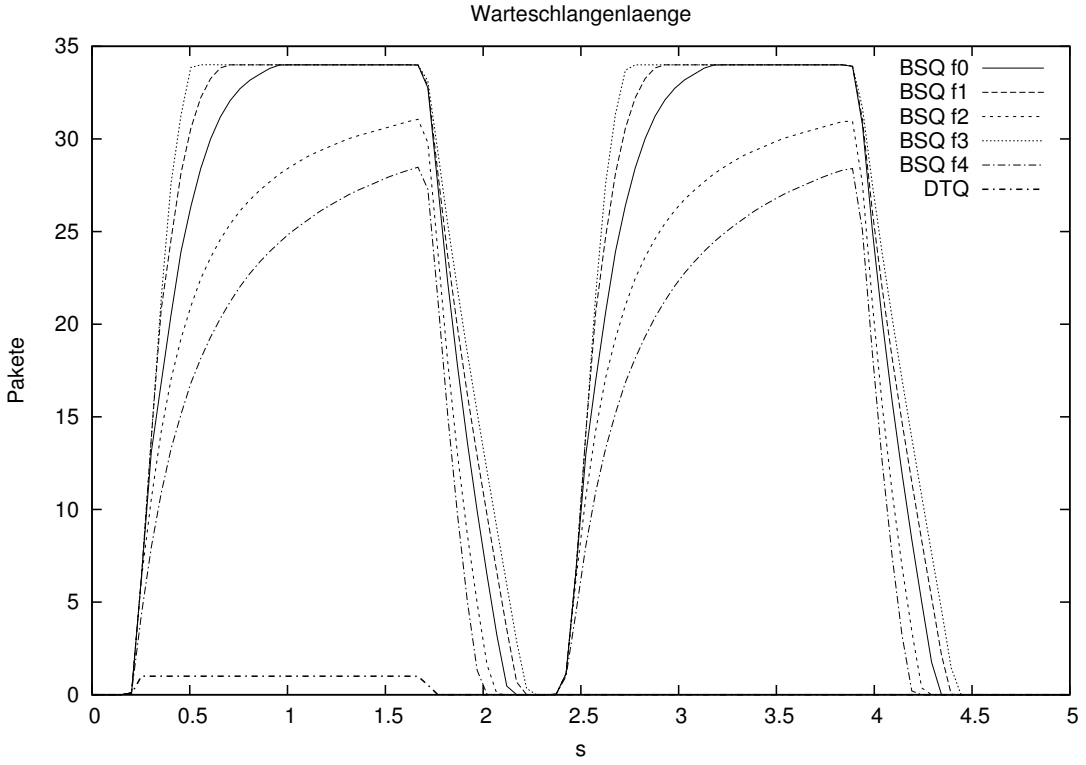

Abbildung B.2.: Warteschlangenlänge Testszenario 2a

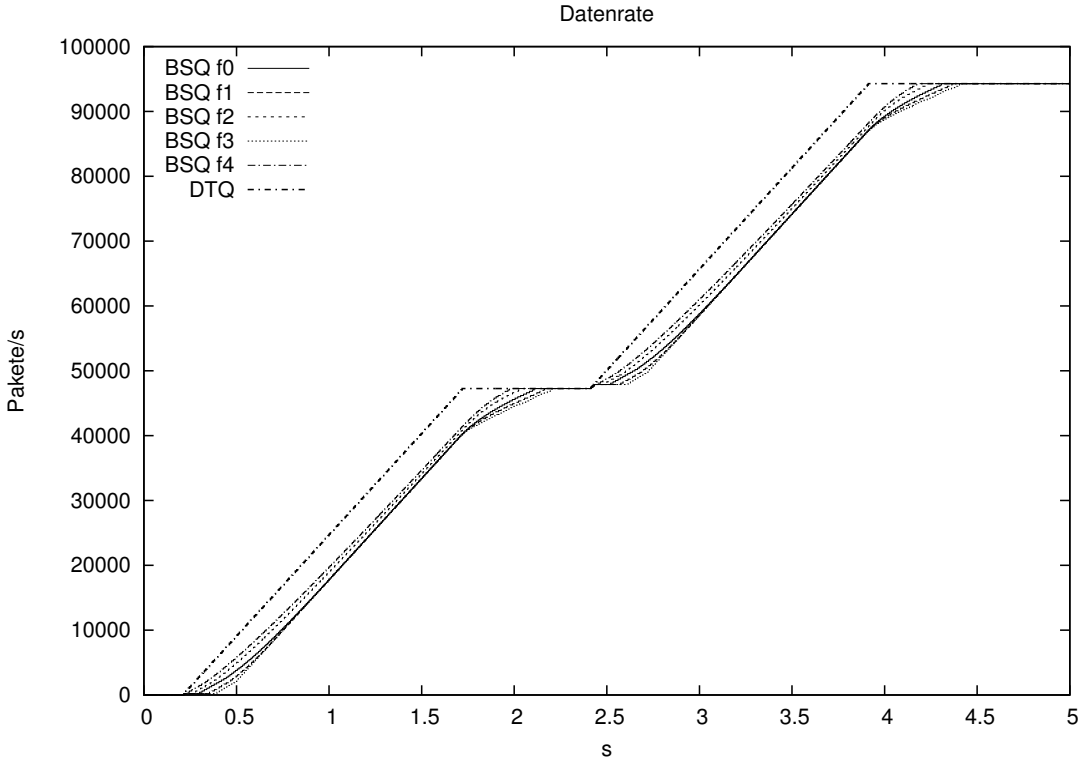

Abbildung B.3.: Datenrate Testszenario 2a

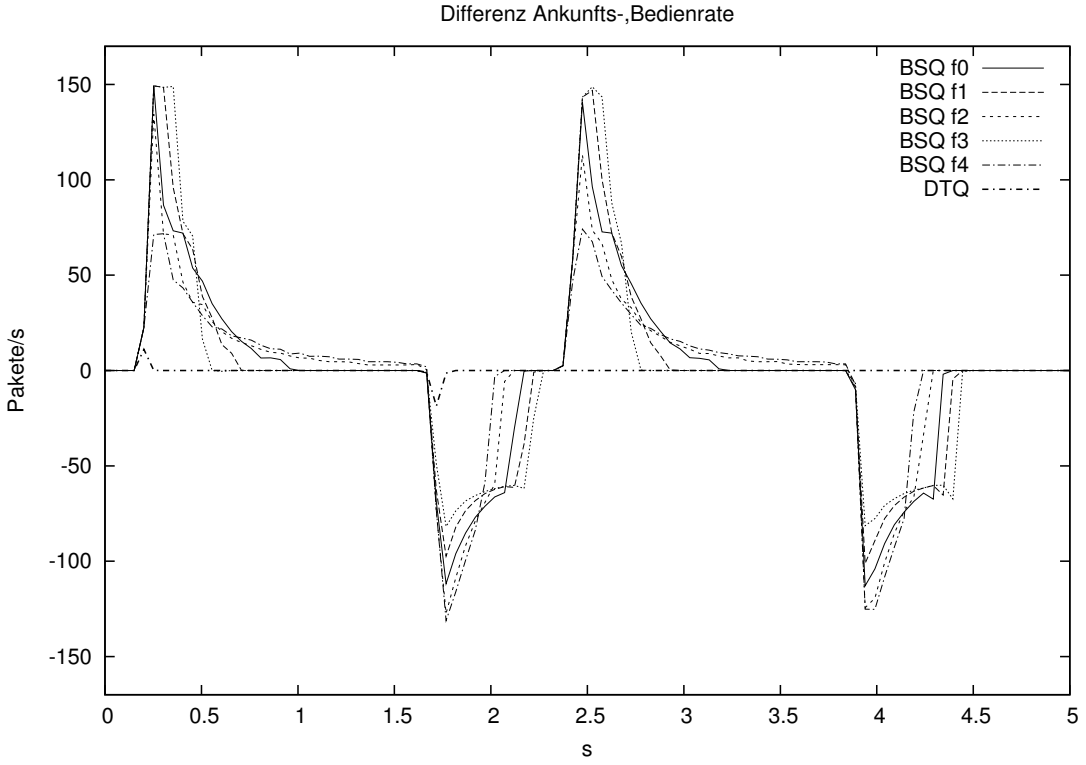

Abbildung B.4.: Differenz Ankunfts-/ Bedienrate Testszenario 2a

C. Messdatendiagramme Testszenario 2b

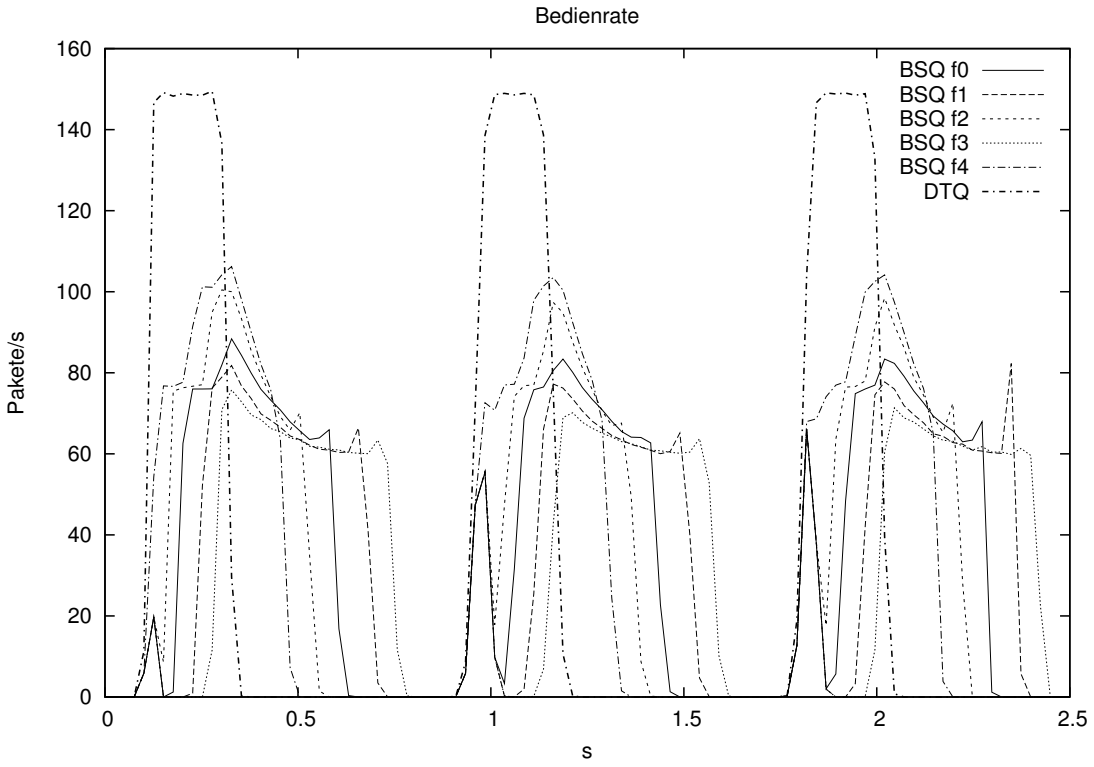

Abbildung C.1.: Bedienraten Testszenario 2b

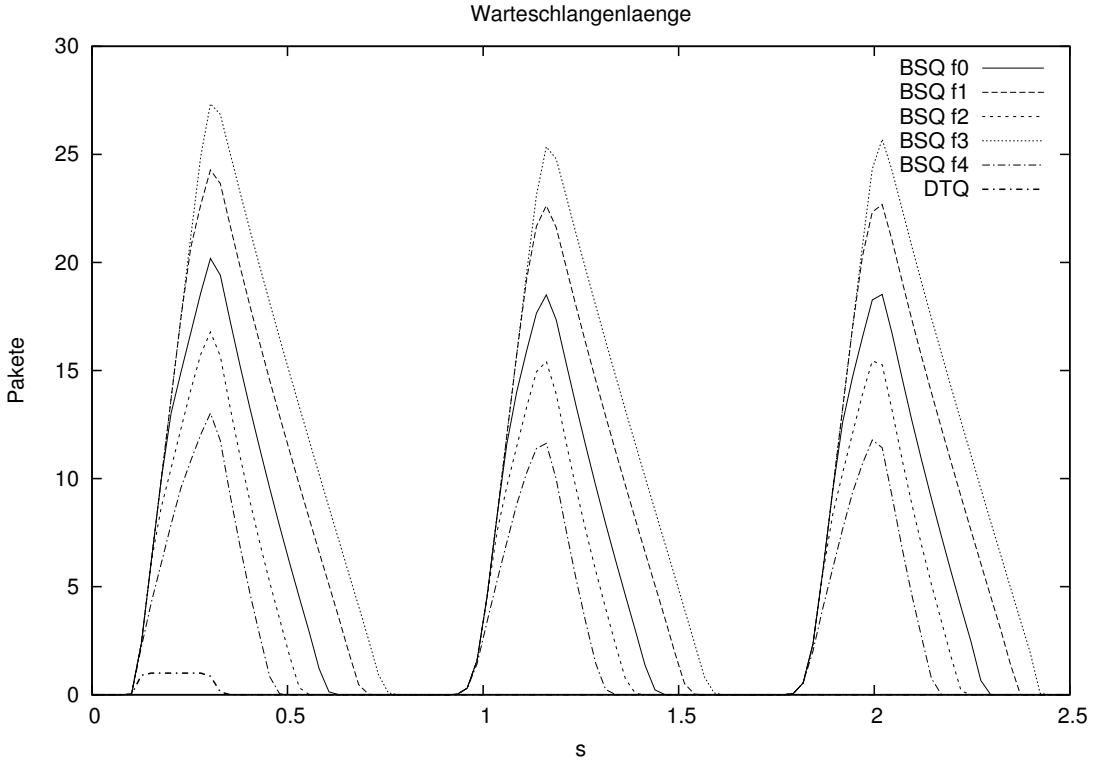

Abbildung C.2.: Warteschlangenlänge Testszenario 2b

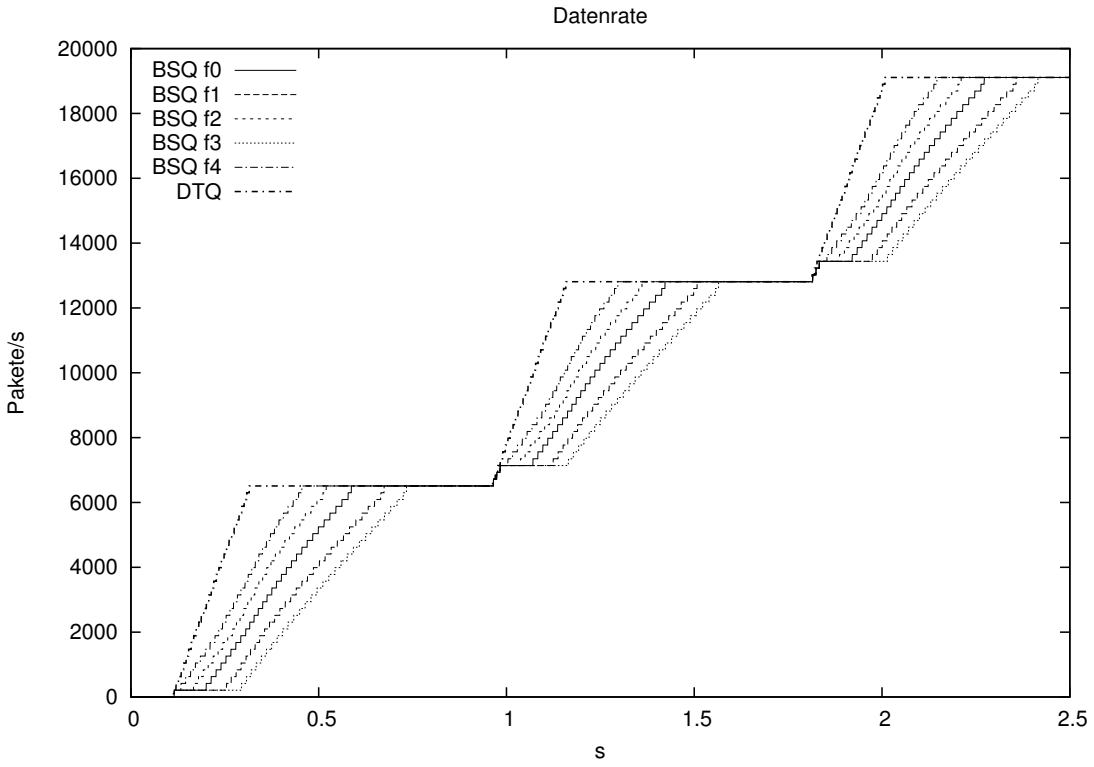

Abbildung C.3.: Datenrate Testszenario 2b

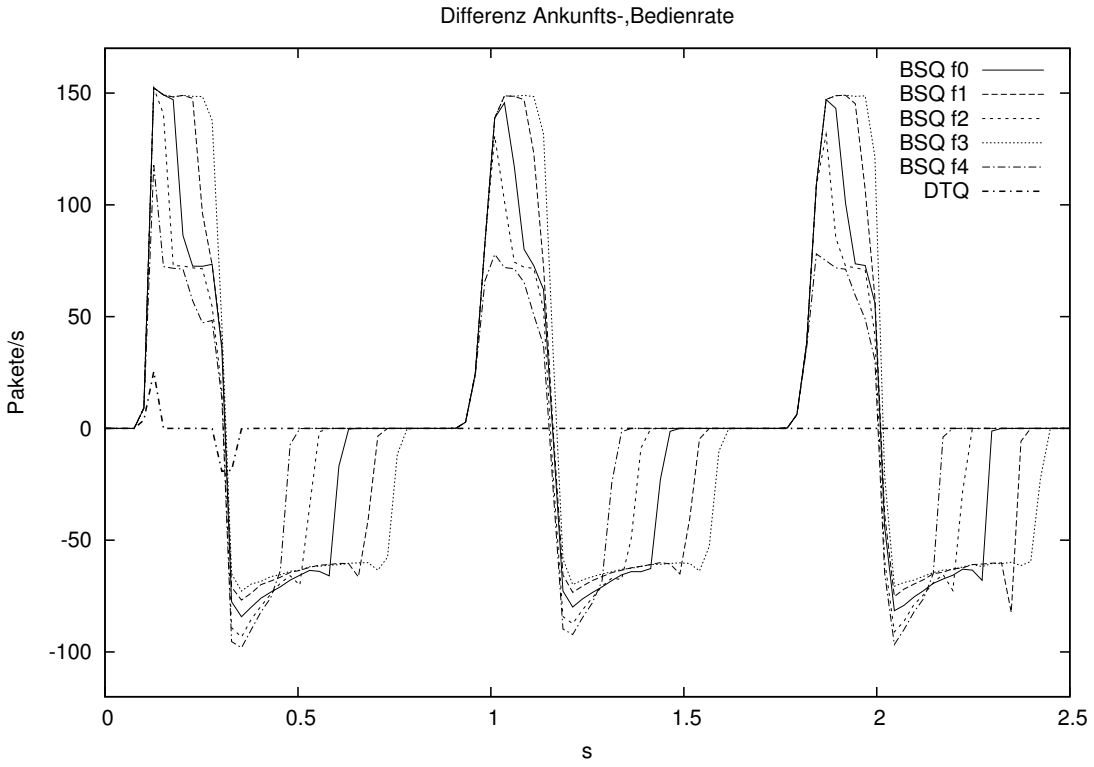

Abbildung C.4.: Differenz Ankunfts-/ Bedienrate Testszenario 2b

D. Messdatendiagramme Testszenario 2c

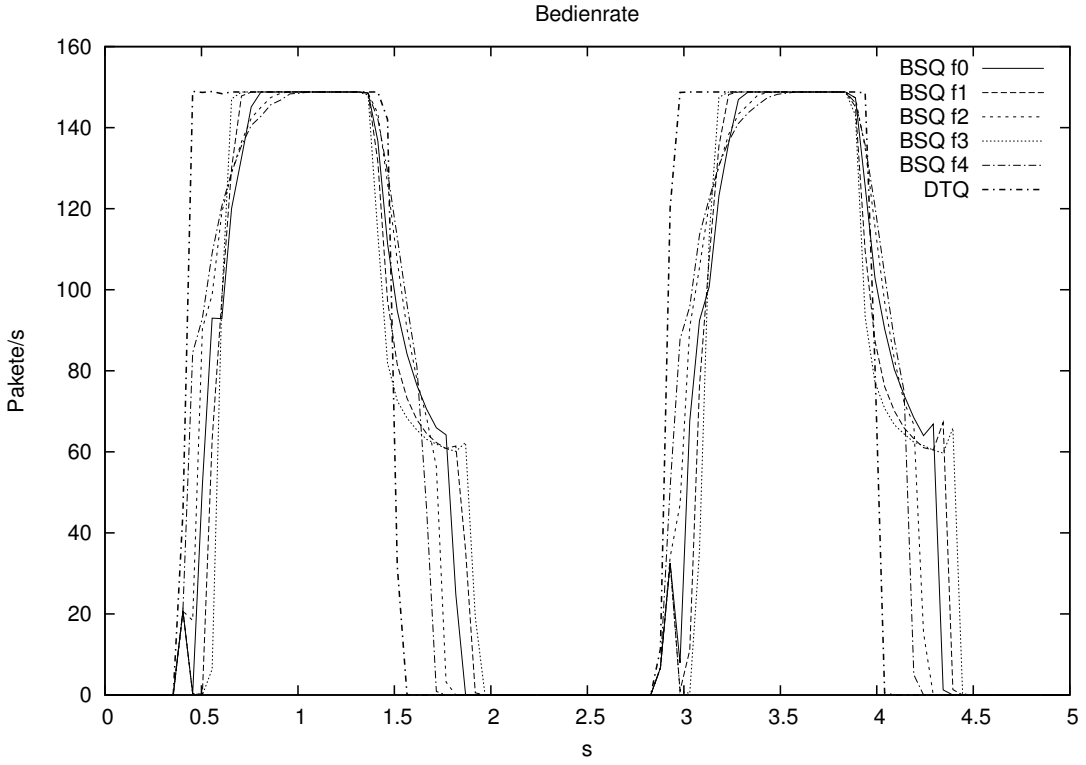

Abbildung D.1.: Bedienraten Testszenario 2c

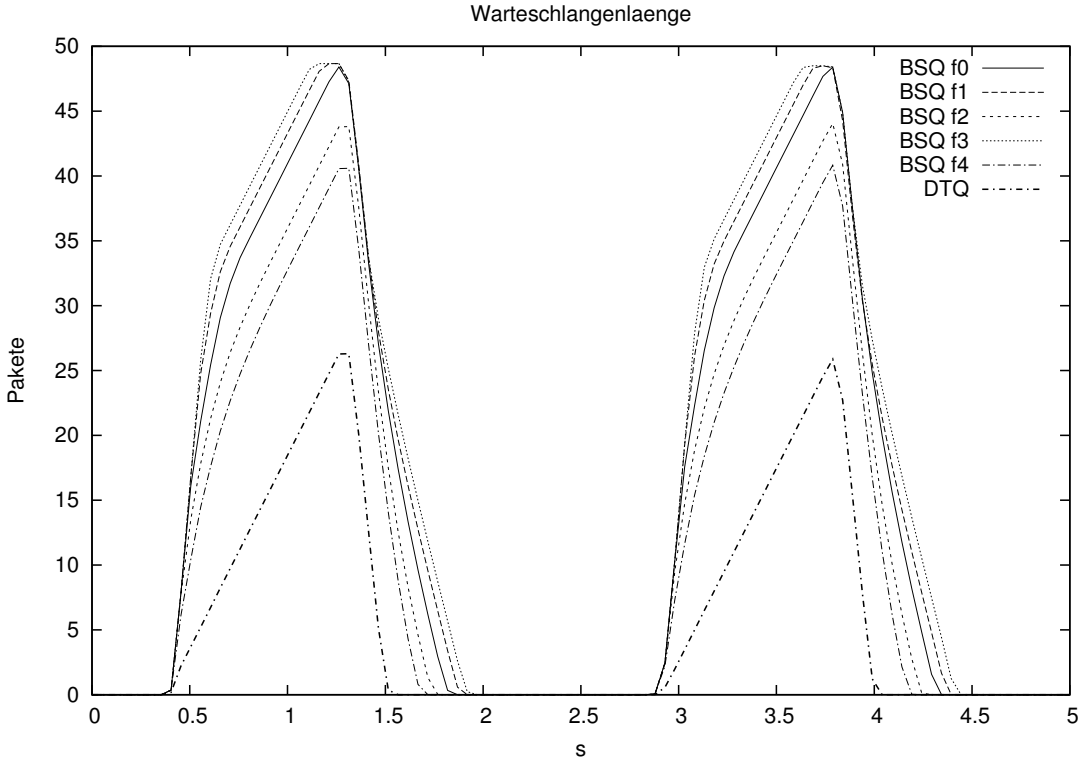

Abbildung D.2.: Warteschlangenlänge Testszenario 2c

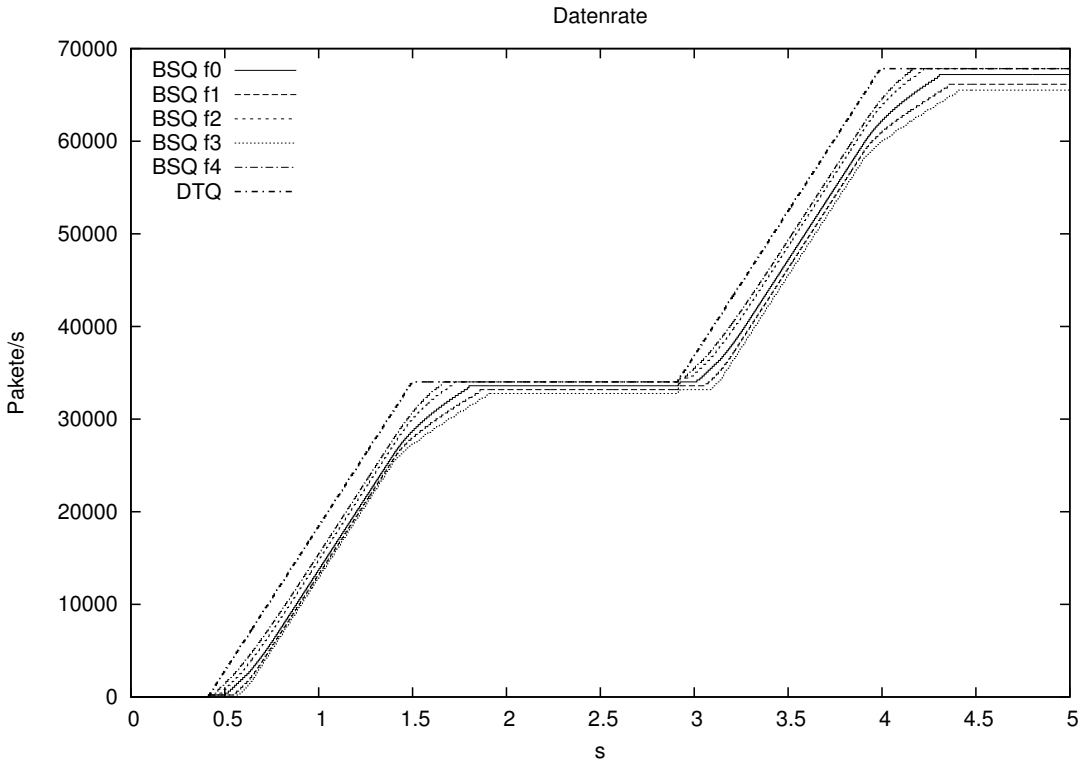

Abbildung D.3.: Datenrate Testszenario 2c

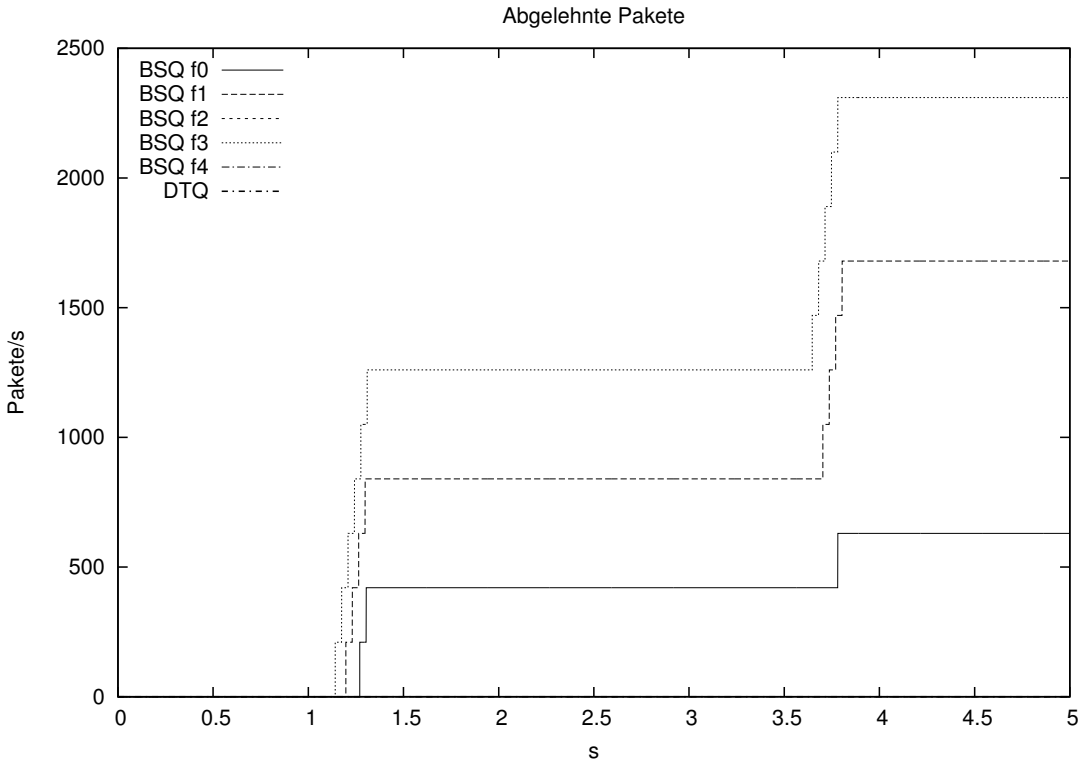

Abbildung D.4.: Abgelehnte Pakete Testszenario 2c

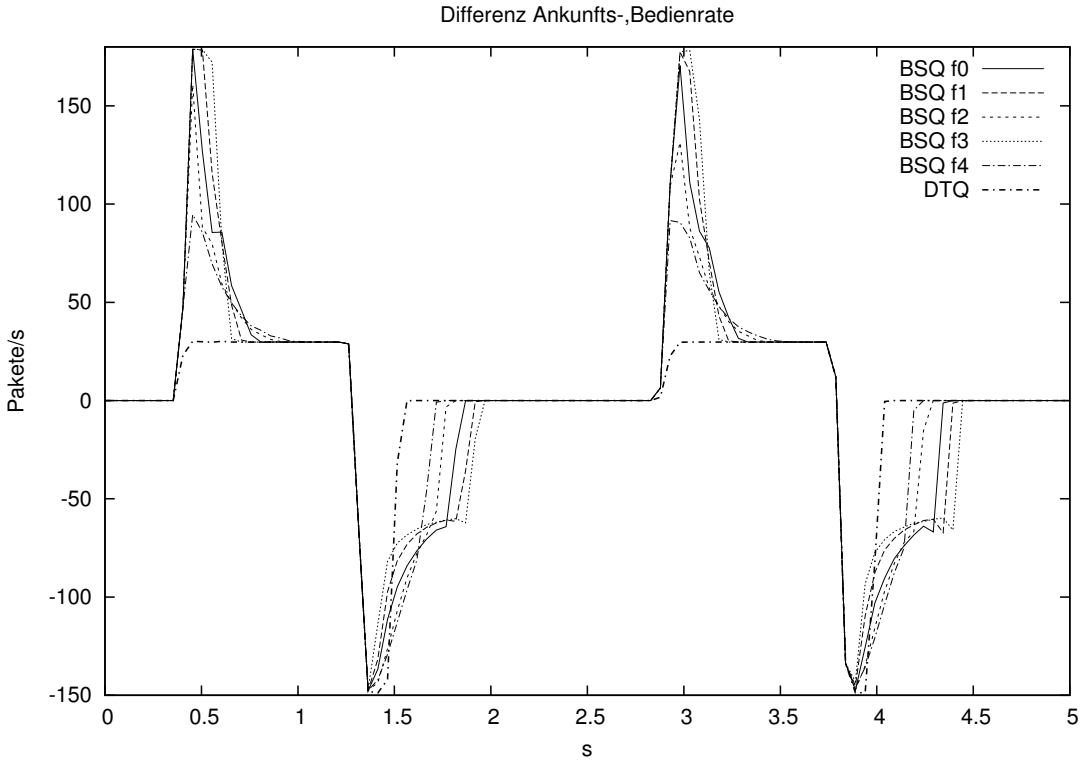

Abbildung D.5.: Differenz Ankunfts-/ Bedienrate Testszenario 2c

E. Erklärung

Ich versichere, dass ich die hier vorliegende Arbeit selbstständig und nur unter der Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 31. Juli 2003

Jens Geier